CHAPTER

Computer Clusters for Scalable Parallel Computing

2

CHAPTER OUTLINE

SUMMARY

Clustering of computers enables scalable parallel and distributed computing in both science and business applications. This chapter is devoted to building cluster-structured massively parallel processors. We focus on the design principles and assessment of the hardware, software, middleware, and operating system support to achieve scalability, availability, programmability, single-system images, and fault tolerance in clusters. We will examine the cluster architectures of Tianhe-1A, Cray XT5 Jaguar, and IBM Roadrunner. The study also covers the LSF middleware and MOSIX/OS for job and resource management in Linux clusters, GPU clusters and cluster extensions to building grids and clouds. Only physical clusters are studied in this chapter. Virtual clusters will be studied in Chapters 3 and 4.

2.1 CLUSTERING FOR MASSIVE PARALLELISM

A *computer cluster* is a collection of interconnected stand-alone computers which can work together collectively and cooperatively as a single integrated computing resource pool. Clustering explores massive parallelism at the job level and achieves *high availability* (HA) through stand-alone operations. The benefits of computer clusters and *massively parallel processors* (MPPs) include scalable performance, HA, fault tolerance, modular growth, and use of commodity components. These features can sustain the generation changes experienced in hardware, software, and network components. Cluster computing became popular in the mid-1990s as traditional mainframes and vector supercomputers were proven to be less cost-effective in many *high-performance computing* (HPC) applications.

Of the Top 500 supercomputers reported in 2010, 85 percent were computer clusters or MPPs built with homogeneous nodes. Computer clusters have laid the foundation for today's supercomputers, computational grids, and Internet clouds built over data centers. We have come a long way toward becoming addicted to computers. According to a recent IDC prediction, the HPC market will increase from $8.5 billion in 2010 to $10.5 billion by 2013. A majority of the Top 500 supercomputers are used for HPC applications in science and engineering. Meanwhile, the use of *high-throughput computing* (HTC) clusters of servers is growing rapidly in business and web services applications.

2.1.1 Cluster Development Trends

Support for clustering of computers has moved from interconnecting high-end mainframe computers to building clusters with massive numbers of x86 engines. Computer clustering started with the linking of large mainframe computers such as the IBM Sysplex and the SGI Origin 3000. Originally, this was motivated by a demand for cooperative group computing and to provide higher availability in critical enterprise applications. Subsequently, the clustering trend moved toward the networking of many minicomputers, such as DEC's VMS cluster, in which multiple VAXes were interconnected to share the same set of disk/tape controllers. Tandem's Himalaya was designed as a business cluster for fault-tolerant *online transaction processing* (OLTP) applications.

In the early 1990s, the next move was to build UNIX-based workstation clusters represented by the Berkeley NOW (Network of Workstations) and IBM SP2 AIX-based server cluster. Beyond

2000, we see the trend moving to the clustering of RISC or x86 PC engines. Clustered products now appear as integrated systems, software tools, availability infrastructure, and operating system extensions. This clustering trend matches the downsizing trend in the computer industry. Supporting clusters of smaller nodes will increase sales by allowing modular incremental growth in cluster configurations. From IBM, DEC, Sun, and SGI to Compaq and Dell, the computer industry has leveraged clustering of low-cost servers or x86 desktops for their cost-effectiveness, scalability, and HA features.

2.1.1.1 Milestone Cluster Systems

Clustering has been a hot research challenge in computer architecture. Fast communication, job scheduling, SSI, and HA are active areas in cluster research. Table 2.1 lists some milestone cluster research projects and commercial cluster products. Details of these old clusters can be found in [14]. These milestone projects have pioneered clustering hardware and middleware development over the past two decades. Each cluster project listed has developed some unique features. Modern clusters are headed toward HPC clusters as studied in Section 2.5.

The NOW project addresses a whole spectrum of cluster computing issues, including architecture, software support for web servers, single system image, I/O and file system, efficient communication, and enhanced availability. The Rice University TreadMarks is a good example of software-implemented shared-memory cluster of workstations. The memory sharing is implemented with a user-space runtime library. This was a research cluster built over Sun Solaris workstations. Some cluster OS functions were developed, but were never marketed successfully.

Table 2.1 Milestone Research or Commercial Cluster Computer Systems [14]

Project	Special Features That Support Clustering
DEC VAXcluster (1991)	A UNIX cluster of symmetric multiprocessing (SMP) servers running the VMS OS with extensions, mainly used in HA applications
U.C. Berkeley NOW Project (1995)	A serverless network of workstations featuring active messaging, cooperative filing, and GLUnix development
Rice University TreadMarks (1996)	Software-implemented distributed shared memory for use in clusters of UNIX workstations based on page migration
Sun Solaris MC Cluster (1995)	A research cluster built over Sun Solaris workstations; some cluster OS functions were developed but were never marketed successfully
Tandem Himalaya Cluster (1994)	A scalable and fault-tolerant cluster for OLTP and database processing, built with nonstop operating system support
IBM SP2 Server Cluster (1996)	An AIX server cluster built with Power2 nodes and the Omega network, and supported by IBM LoadLeveler and MPI extensions
Google Search Engine Cluster (2003)	A 4,000-node server cluster built for Internet search and web service applications, supported by a distributed file system and fault tolerance
MOSIX (2010) www.mosix.org	A distributed operating system for use in Linux clusters, multiclusters, grids, and clouds; used by the research community

A Unix cluster of SMP servers running VMS/OS with extensions, mainly used in high-availability applications. An AIX server cluster built with Power2 nodes and Omega network and supported by IBM Loadleveler and MPI extensions. A scalable and fault-tolerant cluster for OLTP and database processing built with non-stop operating system support. The Google search engine was built at Google using commodity components. MOSIX is a distributed operating systems for use in Linux clusters, multi-clusters, grids, and the clouds, originally developed by Hebrew University in 1999.

2.1.2 Design Objectives of Computer Clusters

Clusters have been classified in various ways in the literature. We classify clusters using six orthogonal attributes: *scalability*, *packaging*, *control*, *homogeneity*, *programmability*, and *security*.

2.1.2.1 Scalability

Clustering of computers is based on the concept of modular growth. To scale a cluster from hundreds of uniprocessor nodes to a supercluster with 10,000 multicore nodes is a nontrivial task. The scalability could be limited by a number of factors, such as the multicore chip technology, cluster topology, packaging method, power consumption, and cooling scheme applied. The purpose is to achieve scalable performance constrained by the aforementioned factors. We have to also consider other limiting factors such as the memory wall, disk I/O bottlenecks, and latency tolerance, among others.

2.1.2.2 Packaging

Cluster nodes can be packaged in a compact or a slack fashion. In a *compact* cluster, the nodes are closely packaged in one or more racks sitting in a room, and the nodes are not attached to peripherals (monitors, keyboards, mice, etc.). In a *slack* cluster, the nodes are attached to their usual peripherals (i.e., they are complete SMPs, workstations, and PCs), and they may be located in different rooms, different buildings, or even remote regions. Packaging directly affects communication wire length, and thus the selection of interconnection technology used. While a compact cluster can utilize a high-bandwidth, low-latency communication network that is often proprietary, nodes of a slack cluster are normally connected through standard LANs or WANs.

2.1.2.3 Control

A cluster can be either controlled or managed in a *centralized* or *decentralized* fashion. A compact cluster normally has centralized control, while a slack cluster can be controlled either way. In a centralized cluster, all the nodes are owned, controlled, managed, and administered by a central operator. In a decentralized cluster, the nodes have individual owners. For instance, consider a cluster comprising an interconnected set of desktop workstations in a department, where each workstation is individually owned by an employee. The owner can reconfigure, upgrade, or even shut down the workstation at any time. This lack of a single point of control makes system administration of such a cluster very difficult. It also calls for special techniques for process scheduling, workload migration, checkpointing, accounting, and other similar tasks.

2.1.2.4 Homogeneity

A *homogeneous* cluster uses nodes from the same platform, that is, the same processor architecture and the same operating system; often, the nodes are from the same vendors. A *heterogeneous*

cluster uses nodes of different platforms. Interoperability is an important issue in heterogeneous clusters. For instance, process migration is often needed for load balancing or availability. In a homogeneous cluster, a binary process image can migrate to another node and continue execution. This is not feasible in a heterogeneous cluster, as the binary code will not be executable when the process migrates to a node of a different platform.

2.1.2.5 Security

Intracluster communication can be either *exposed* or *enclosed*. In an exposed cluster, the communication paths among the nodes are exposed to the outside world. An outside machine can access the communication paths, and thus individual nodes, using standard protocols (e.g., TCP/IP). Such exposed clusters are easy to implement, but have several disadvantages:

- Being exposed, intracluster communication is not secure, unless the communication subsystem performs additional work to ensure privacy and security.
- Outside communications may disrupt intracluster communications in an unpredictable fashion. For instance, heavy BBS traffic may disrupt production jobs.
- Standard communication protocols tend to have high overhead.

In an enclosed cluster, intracluster communication is shielded from the outside world, which alleviates the aforementioned problems. A disadvantage is that there is currently no standard for efficient, enclosed intracluster communication. Consequently, most commercial or academic clusters realize fast communications through one-of-a-kind protocols.

2.1.2.6 Dedicated versus Enterprise Clusters

A *dedicated cluster* is typically installed in a deskside rack in a central computer room. It is homogeneously configured with the same type of computer nodes and managed by a single administrator group like a frontend host. Dedicated clusters are used as substitutes for traditional mainframes or supercomputers. A dedicated cluster is installed, used, and administered as a single machine. Many users can log in to the cluster to execute both interactive and batch jobs. The cluster offers much enhanced throughput, as well as reduced response time.

An *enterprise cluster* is mainly used to utilize idle resources in the nodes. Each node is usually a full-fledged SMP, workstation, or PC, with all the necessary peripherals attached. The nodes are typically geographically distributed, and are not necessarily in the same room or even in the same building. The nodes are individually owned by multiple owners. The cluster administrator has only limited control over the nodes, as a node can be turned off at any time by its owner. The owner's "local" jobs have higher priority than enterprise jobs. The cluster is often configured with heterogeneous computer nodes. The nodes are often connected through a low-cost Ethernet network. Most data centers are structured with clusters of low-cost servers. Virtual clusters play a crucial role in upgrading data centers. We will discuss virtual clusters in Chapter 6 and clouds in Chapters 7, 8, and 9.

2.1.3 Fundamental Cluster Design Issues

In this section, we will classify various cluster and MPP families. Then we will identify the major design issues of clustered and MPP systems. Both physical and virtual clusters are covered. These systems are often found in computational grids, national laboratories, business data centers,

supercomputer sites, and virtualized cloud platforms. A good understanding of how clusters and MPPs work collectively will pave the way toward understanding the ins and outs of large-scale grids and Internet clouds in subsequent chapters. Several issues must be considered in developing and using a cluster. Although much work has been done in this regard, this is still an active research and development area.

2.1.3.1 Scalable Performance

This refers to the fact that scaling of resources (cluster nodes, memory capacity, I/O bandwidth, etc.) leads to a proportional increase in performance. Of course, both scale-up and scale-down capabilities are needed, depending on application demand or cost-effectiveness considerations. Clustering is driven by scalability. One should not ignore this factor in all applications of cluster or MPP computing systems.

2.1.3.2 Single-System Image (SSI)

A set of workstations connected by an Ethernet network is not necessarily a cluster. A cluster is a single system. For example, suppose a workstation has a 300 Mflops/second processor, 512 MB of memory, and a 4 GB disk and can support 50 active users and 1,000 processes. By clustering 100 such workstations, can we get a single system that is equivalent to one huge workstation, or a *megastation*, that has a 30 Gflops/second processor, 50 GB of memory, and a 400 GB disk and can support 5,000 active users and 100,000 processes? This is an appealing goal, but it is very difficult to achieve. SSI techniques are aimed at achieving this goal.

2.1.3.3 Availability Support

Clusters can provide cost-effective HA capability with lots of redundancy in processors, memory, disks, I/O devices, networks, and operating system images. However, to realize this potential, availability techniques are required. We will illustrate these techniques later in the book, when we discuss how DEC clusters (Section 10.4) and the IBM SP2 (Section 10.3) attempt to achieve HA.

2.1.3.4 Cluster Job Management

Clusters try to achieve high system utilization from traditional workstations or PC nodes that are normally not highly utilized. Job management software is required to provide batching, load balancing, parallel processing, and other functionality. We will study cluster job management systems in Section 3.4. Special software tools are needed to manage multiple jobs simultaneously.

2.1.3.5 Internode Communication

Because of their higher node complexity, cluster nodes cannot be packaged as compactly as MPP nodes. The internode physical wire lengths are longer in a cluster than in an MPP. This is true even for centralized clusters. A long wire implies greater interconnect network latency. But more importantly, longer wires have more problems in terms of reliability, clock skew, and cross talking. These problems call for reliable and secure communication protocols, which increase overhead. Clusters often use commodity networks (e.g., Ethernet) with standard protocols such as TCP/IP.

2.1.3.6 Fault Tolerance and Recovery

Clusters of machines can be designed to eliminate all single points of failure. Through redundancy, a cluster can tolerate faulty conditions up to a certain extent. Heartbeat mechanisms can be installed

to monitor the running condition of all nodes. In case of a node failure, critical jobs running on the failing nodes can be saved by failing over to the surviving node machines. Rollback recovery schemes restore the computing results through periodic checkpointing.

2.1.3.7 Cluster Family Classification

Based on application demand, computer clusters are divided into three classes:

- **Compute clusters** These are clusters designed mainly for collective computation over a single large job. A good example is a cluster dedicated to numerical simulation of weather conditions. The compute clusters do not handle many I/O operations, such as database services. When a single compute job requires frequent communication among the cluster nodes, the cluster must share a dedicated network, and thus the nodes are mostly homogeneous and tightly coupled. This type of clusters is also known as a ***Beowulf cluster***.
  When the nodes require internode communication over a small number of heavy-duty nodes, they are essentially known as a *computational grid*. Tightly coupled compute clusters are designed for supercomputing applications. Compute clusters apply middleware such as a message-passing interface (**MPI**) or Parallel Virtual Machine (**PVM**) to port programs to a wide variety of clusters.
- **High-Availability clusters** HA (high-availability) clusters are designed to be fault-tolerant and achieve HA of services. HA clusters operate with many redundant nodes to sustain faults or failures. The simplest HA cluster has only two nodes that can fail over to each other. Of course, high redundancy provides higher availability. HA clusters should be designed to avoid all single points of failure. Many commercial HA clusters are available for various operating systems.
- **Load-balancing clusters** These clusters shoot for higher resource utilization through load balancing among all participating nodes in the cluster. All nodes share the workload or function as a single *virtual machine* (VM). Requests initiated from the user are distributed to all node computers to form a cluster. This results in a balanced workload among different machines, and thus higher resource utilization or higher performance. Middleware is needed to achieve dynamic load balancing by job or process migration among all the cluster nodes.

2.1.4 Analysis of the Top 500 Supercomputers

Every six months, the world's Top 500 supercomputers are evaluated by running the Linpack Benchmark program over very large data sets. The ranking varies from year to year, similar to a competition. In this section, we will analyze the historical share in architecture, speed, operating systems, countries, and applications over time. In addition, we will compare the top five fastest systems in 2010.

2.1.4.1 Architectural Evolution

It is interesting to observe in Figure 2.1 the architectural evolution of the Top 500 supercomputers over the years. In 1993, 250 systems assumed the SMP architecture and these SMP systems all disappeared in June of 2002. Most SMPs are built with shared memory and shared I/O devices. There were 120 MPP systems built in 1993, MPPs reached the peak mof 350 systems in mod-200, and dropped to less than 100 systems in 2010. The single instruction, multiple data (SIMD) machines disappeared in 1997. The cluster architecture appeared in a few systems in 1999. The cluster systems are now populated in the Top-500 list with more than 400 systems as the dominating architecture class.

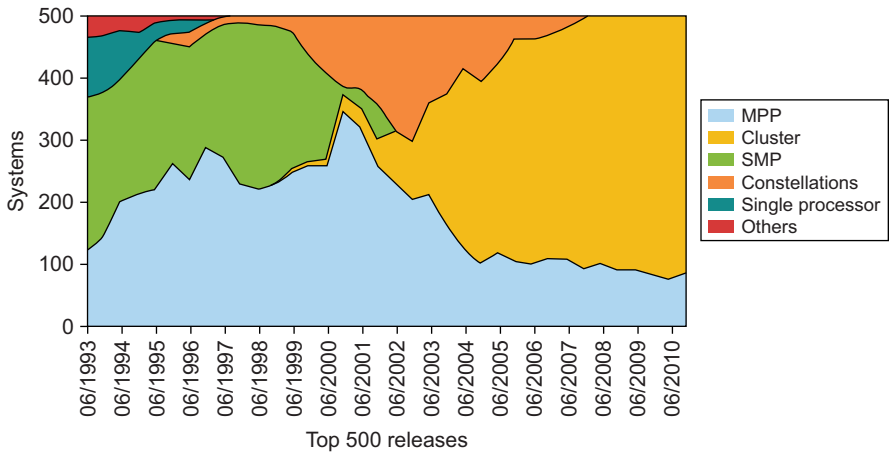


FIGURE 2.1

Architectural share of the Top 500 systems.

(*Courtesy of www.top500.org [25]*)

In 2010, the Top 500 architecture is dominated by clusters (420 systems) and MPPs (80 systems). The basic distinction between these two classes lies in the components they use to build the systems. Clusters are often built with commodity hardware, software, and network components that are commercially available. MPPs are built with custom-designed compute nodes, boards, modules, and cabinets that are interconnected by special packaging. MPPs demand high bandwidth, low latency, better power efficiency, and high reliability. Cost-wise, clusters are affordable by allowing modular growth with scaling capability. The fact that MPPs appear in a much smaller quantity is due to their high cost. Typically, only a few MPP-based supercomputers are installed in each country.

2.1.4.2 Speed Improvement over Time

Figure 2.2 plots the measured performance of the Top 500 fastest computers from 1993 to 2010. The y-axis is scaled by the sustained speed performance in terms of Gflops, Tflops, and Pflops. The middle curve plots the performance of the fastest computers recorded over 17 years; peak performance increases from 58.7 Gflops to 2.566 Pflops. The bottom curve corresponds to the speed of the 500th computer, which increased from 0.42 Gflops in 1993 to 31.1 Tflops in 2010. The top curve plots the speed sum of all 500 computers over the same period. In 2010, the total speed sum of 43.7 Pflops was achieved by all 500 computers, collectively. It is interesting to observe that the total speed sum increases almost linearly with time.

2.1.4.3 Operating System Trends in the Top 500

The five most popular operating systems have more than a 10 percent share among the Top 500 computers, according to data released by TOP500.org (www.top500.org/stats/list/36/os) in November 2010. According to the data, 410 supercomputers are using Linux with a total processor count exceeding 4.5 million. This constitutes 82 percent of the systems adopting Linux. The IBM AIX/OS is in second place with 17 systems (a 3.4 percent share) and more than 94,288 processors.

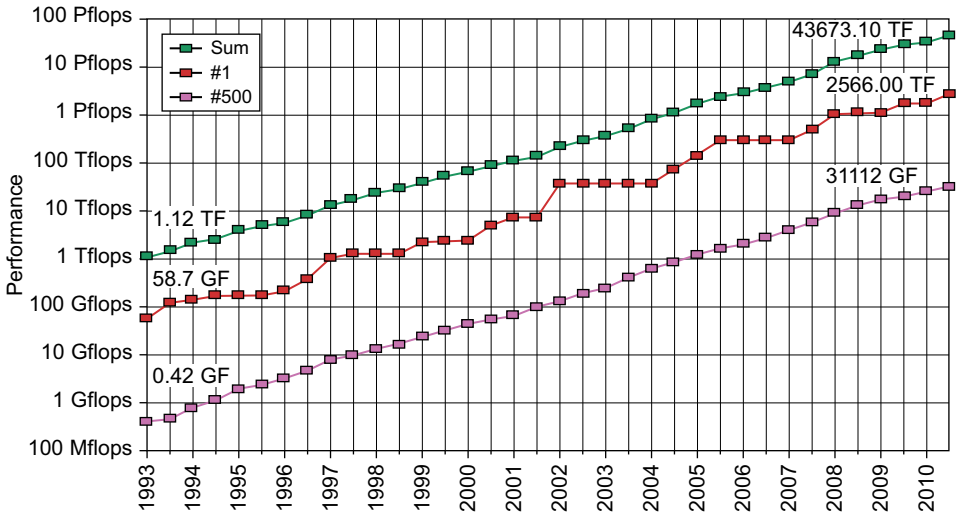


FIGURE 2.2

Performance plot of the Top 500 supercomputers from 1993 to 2010.

(*Courtesy of www.top500.org [25]*)

Third place is represented by the combined use of the SLEs10 with the SGI ProPack5, with 15 systems (3 percent) over 135,200 processors. Fourth place goes to the CNK/SLES9 used by 14 systems (2.8 percent) over 1.13 million processors. Finally, the CNL/OS was used in 10 systems (2 percent) over 178,577 processors. The remaining 34 systems applied 13 other operating systems with a total share of only 6.8 percent. In conclusion, the Linux OS dominates the systems in the Top 500 list.

2.1.4.4 The Top Five Systems in 2010

In Table 2.2, we summarize the key architecture features and sustained Linpack Benchmark performance of the top five supercomputers reported in November 2010. The Tianhe-1A was ranked as the fastest MPP in late 2010. This system was built with 86,386 Intel Xeon CPUs and NVIDIA GPUs by the National University of Defense Technology in China. We will present in Section 2.5 some of the top winners: namely the Tianhe-1A, Cray Jaguar, Nebulae, and IBM Roadrunner that were ranked among the top systems from 2008 to 2010. All the top five machines in Table 2.3 have achieved a speed higher than 1 Pflops. The *sustained speed*, $\boldsymbol{R_{max}}$, in Pflops is measured from the execution of the Linpack Benchmark program corresponding to a maximum matrix size.

The *system efficiency* reflects the ratio of the *sustained speed* to the *peak speed*, $\boldsymbol{R_{peak}}$, when all computing elements are fully utilized in the system. Among the top five systems, the two U.S.-built systems, the Jaguar and the Hopper, have the highest efficiency, more than 75 percent. The two systems built in China, the Tianhe-1A and the Nebulae, and Japan's TSUBAME 2.0, are all low in efficiency. In other words, these systems should still have room for improvement in the future. The average power consumption in these 5 systems is 3.22 MW. This implies that excessive power consumption may post the limit to build even faster supercomputers in the future. These top systems all emphasize massive parallelism using up to 250,000 processor cores per system.

Table 2.2 The Top Five Supercomputers Evaluated in November 2010

System Name, Site, and URL	System Name, Processors, OS, Topology, and Developer	Linpack Speed (R_{max}), Power	Efficiency (R_{max}/R_{peak})
1. Tianhe-1A, National Supercomputing Center, Tianjin, China, http://www.nscc-tj.gov.cn/en/	NUDT TH1A with 14,336 Xeon X5670 CPUs (six cores each) plus 7168 NVIDIA Tesla M2050 GPUs (448 CUDA cores each), running Linux, built by National Univ. of Defense Technology, China	2.57 Pflops, 4.02 MW	54.6% (over a peak of 4.7 Pflops)
2. Jaguar, DOE/SC/Oak Ridge National Lab., United States, http://computing.ornl.gov	Cray XT5-HE: MPP with 224,162 x 6 AMD Opteron, 3D torus network, Linux (CLE), manufactured by Cray, Inc.	1.76 Pflops, 6.95 MW	75.6% (over a peak of 4.7 Pflops)
3. Nebulae at China's National Supercomputer Center, ShenZhen, China http://www.ict.cas.cas.cn	TC3600 Blade, 120,640 cores in 55,680 Xeon X5650 plus 64,960 NVIDIA Tesla C2050 GPUs, Linux, InfiniBand, built by Dawning, Inc.	1.27 Pflops, 2.55 MW	42.6% (over a peak of 2.98 Pflops)
4. TSUBAME 2.0, GSIC Center, Tokyo Institute of Technology, Tokyo, Japan, http://www.gsic.titech.ac.jp/	HP cluster, 3000SL, 73,278 x 6 Xeon X5670 processors, NVIDIA GPU, Linux/SLES 11, built by NEC/HP	1.19 Pflops, 1.8 MW	52.19% (over a peak of 2.3 Pflops)
5. Hopper, DOE/SC/LBNL/ NERSC, Berkeley, CA. USA, http//www/nersc.gov/	Cray XE6 150,408 x 12 AMD Opteron, Linux (CLE), built by Cray, Inc.	1.05 Pflops, 2.8 MW	78.47% (over a peak of 1.35 Pflops)

Table 2.3 Sample Compute Node Architectures for Large Cluster Construction

Node Architecture	Major Characteristics	Representative Systems
Homogeneous node using the same multicore processors	Multicore processors mounted on the same node with a crossbar connected to shared memory or local disks	The Cray XT5 uses two six-core AMD Opteron processors in each compute node
Hybrid nodes using CPU plus GPU or FLP accelerators	General-purpose CPU for integer operations, with GPUs acting as coprocessors to speed up FLP operations	China's Tianhe-1A uses two Intel Xeon processors plus one NVIDIA GPU per compute node

2.1.4.5 Country Share and Application Share

In the 2010 Top-500 list, there were 274 supercomputing systems installed in the US, 41 systems in China, and 103 systems in Japan, France, UK, and Germany. The remaining countries have 82 systems. This country share roughly reflects the countries' economic growth over the years. Countries compete in the Top 500 race every six months. Major increases of supercomputer applications are in the areas of database, research, finance, and information services.

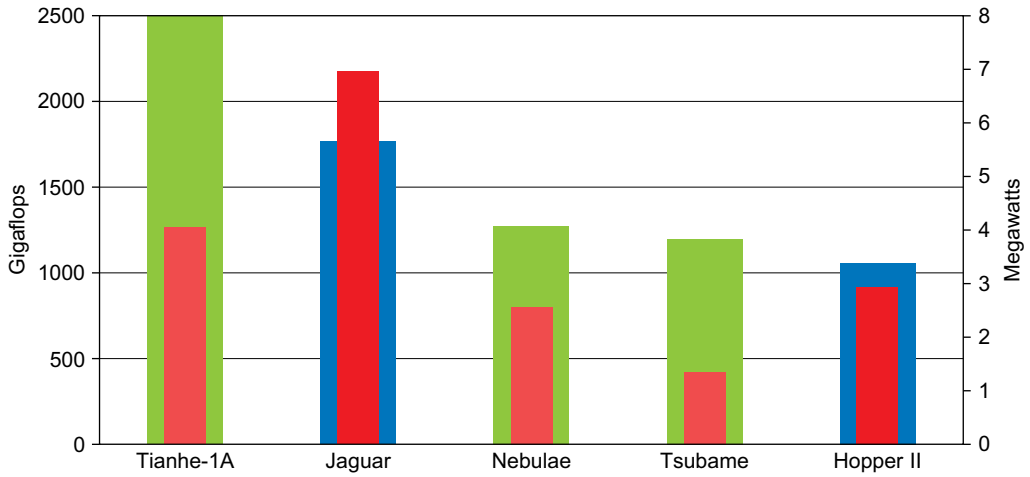


FIGURE 2.3

Power and performance of the top 5 supercomputers in November 2010.

(*Courtesy of www.top500.org [25] and B. Dally [10]*)

2.1.4.6 Power versus Performance in the Top Five in 2010

In Figure 2.3, the top five supercomputers are ranked by their speed (Gflops) on the left side and by their power consumption (MW per system) on the right side. The Tiahhe-1A scored the highest with a 2.57 Pflops speed and 4.01 MW power consumption. In second place, the Jaguar consumes the highest power of 6.9 MW. In fourth place, the TSUBAME system consumes the least power, 1.5 MW, and has a speed performance that almost matches that of the Nebulae system. One can define a performance/power ratio to see the trade-off between these two metrics. There is also a Top 500 Green contest that ranks the supercomputers by their power efficiency. This chart shows that all systems using the hybrid CPU/GPU architecture consume much less power.

2.2 COMPUTER CLUSTERS AND MPP ARCHITECTURES

Most clusters emphasize higher availability and scalable performance. Clustered systems evolved from the Microsoft Wolfpack and Berkeley NOW to the SGI Altix Series, IBM SP Series, and IBM Roadrunner. NOW was a UC Berkeley research project designed to explore new mechanisms for clustering of UNIX workstations. Most clusters use commodity networks such as Gigabit Ethernet, Myrinet switches, or InfiniBand networks to interconnect the compute and storage nodes. The clustering trend moves from supporting large rack-size, high-end computer systems to high-volume, desktop or deskside computer systems, matching the downsizing trend in the computer industry.

2.2.1 Cluster Organization and Resource Sharing

In this section, we will start by discussing basic, small-scale PC or server clusters. We will discuss how to construct large-scale clusters and MPPs in subsequent sections.

2.2.1.1 A Basic Cluster Architecture

Figure 2.4 shows the basic architecture of a computer cluster over PCs or workstations. The figure shows a simple cluster of computers built with commodity components and fully supported with desired SSI features and HA capability. The processing nodes are commodity workstations, PCs, or servers. These commodity nodes are easy to replace or upgrade with new generations of hardware. The node operating systems should be designed for multiuser, multitasking, and multithreaded applications. The nodes are interconnected by one or more fast commodity networks. These networks use standard communication protocols and operate at a speed that should be two orders of magnitude faster than that of the current TCP/IP speed over Ethernet.

The network interface card is connected to the node's standard I/O bus (e.g., PCI). When the processor or the operating system is changed, only the driver software needs to change. We desire to have a platform-independent *cluster operating system*, sitting on top of the node platforms. But such a cluster OS is not commercially available. Instead, we can deploy some cluster middleware to glue together all node platforms at the user space. An availability middleware offers HA services. An SSI layer provides a single entry point, a single file hierarchy, a single point of control, and a

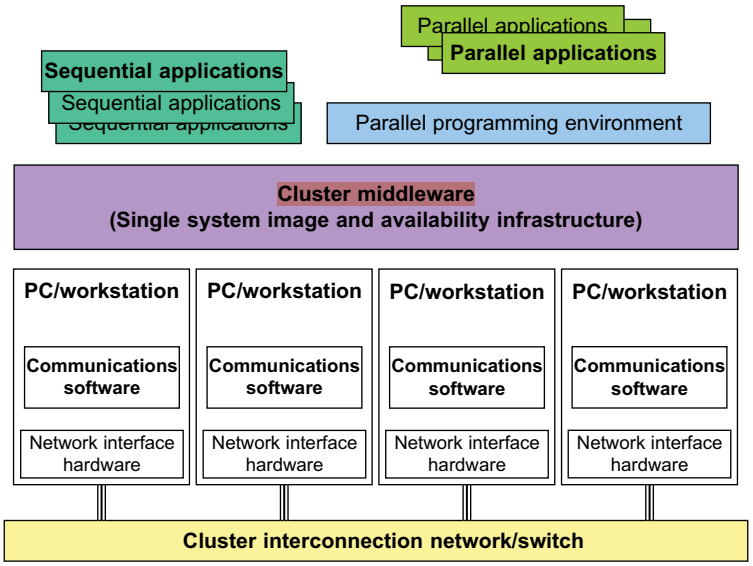


FIGURE 2.4

The architecture of a computer cluster built with commodity hardware, software, middleware, and network components supporting HA and SSI.

(*Courtesy of M. Baker and R. Buyya, reprinted with Permission [3]*)

single job management system. Single memory may be realized with the help of the compiler or a runtime library. A single process space is not necessarily supported.

In general, an idealized cluster is supported by three subsystems. First, conventional databases and OLTP monitors offer users a desktop environment in which to use the cluster. In addition to running sequential user programs, the cluster supports parallel programming based on standard languages and communication libraries using PVM, MPI, or OpenMP. The programming environment also includes tools for debugging, profiling, monitoring, and so forth. A user interface subsystem is needed to combine the advantages of the web interface and the Windows GUI. It should also provide user-friendly links to various programming environments, job management tools, hypertext, and search support so that users can easily get help in programming the computer cluster.

2.2.1.2 Resource Sharing in Clusters

Supporting clusters of smaller nodes will increase computer sales. Clustering improves both availability and performance. These two clustering goals are not necessarily in conflict. Some HA clusters use hardware redundancy for scalable performance. The nodes of a cluster can be connected in one of three ways, as shown in Figure 2.5. The shared-nothing architecture is used in most clusters, where the nodes are connected through the I/O bus. The shared-disk architecture is in favor of small-scale *availability clusters* in business applications. When one node fails, the other node takes over.

The shared-nothing configuration in Part (a) simply connects two or more autonomous computers via a LAN such as Ethernet. A shared-disk cluster is shown in Part (b). This is what most business clusters desire so that they can enable recovery support in case of node failure. The shared disk can hold checkpoint files or critical system images to enhance cluster availability. Without shared disks, checkpointing, rollback recovery, failover, and failback are not possible in a cluster. The shared-memory cluster in Part (c) is much more difficult to realize. The nodes could be connected by a *scalable coherence interface (SCI)* ring, which is connected to the memory bus of each node through an NIC module. In the other two architectures, the interconnect is attached to the I/O bus. The memory bus operates at a higher frequency than the I/O bus.

There is no widely accepted standard for the memory bus. But there are such standards for the I/O buses. One recent, popular standard is the PCI I/O bus standard. So, if you implement an NIC card to attach a faster Ethernet network to the PCI bus you can be assured that this card can be used in other systems that use PCI as the I/O bus. The I/O bus evolves at a much slower rate than the memory bus. Consider a cluster that uses connections through the PCI bus. When the processors are upgraded, the interconnect and the NIC do not have to change, as long as the new system still uses PCI. In a shared-memory cluster, changing the processor implies a redesign of the node board and the NIC card.

2.2.2 Node Architectures and MPP Packaging

In building large-scale clusters or MPP systems, cluster nodes are classified into two categories: *compute nodes* and *service nodes*. Compute nodes appear in larger quantities mainly used for large-scale searching or parallel floating-point computations. Service nodes could be built with different processors mainly used to handle I/O, file access, and system monitoring. For MPP clusters, the

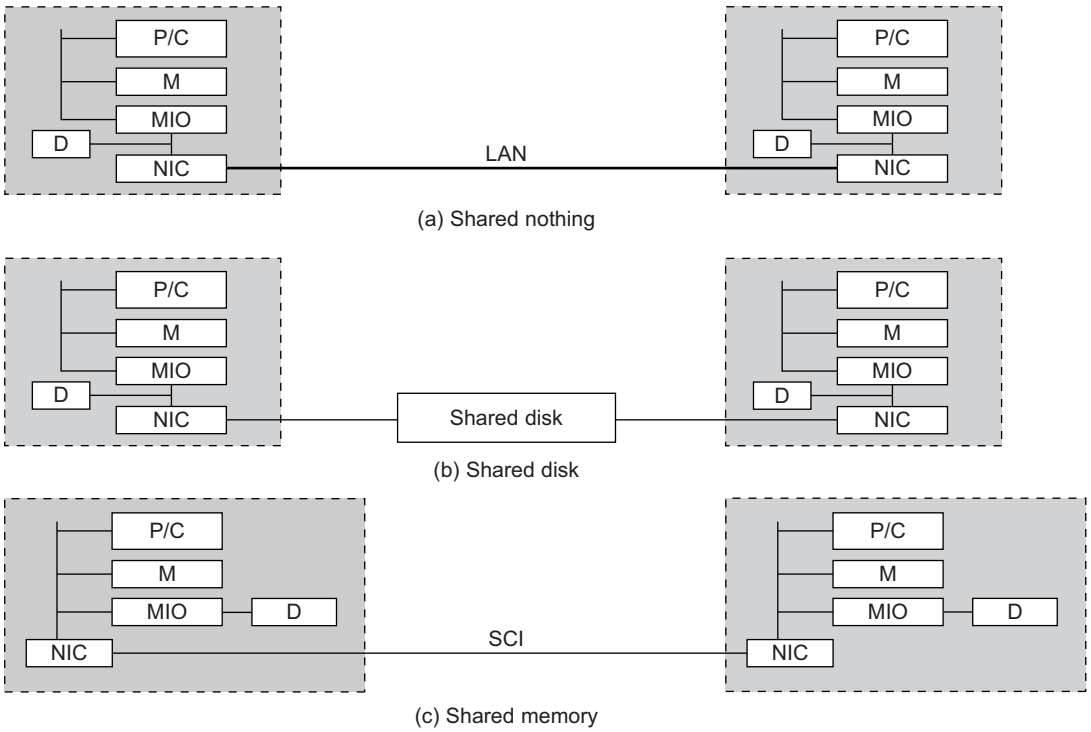


FIGURE 2.5

Three ways to connect cluster nodes (P/C: Processor and Cache; M: Memory; D: Disk; NIC: Network Interface Circuitry; MIO: Memory-I/O Bridge.)

(*Courtesy of Hwang and Xu [14]*)

compute nodes dominate in system cost, because we may have 1,000 times more compute nodes than service nodes in a single large clustered system. Table 2.3 introduces two example compute node architectures: *homogeneous design* and *hybrid node design.*

In the past, most MPPs are built with a homogeneous architecture by interconnecting a large number of the same compute nodes. In 2010, the Cray XT5 Jaguar system was built with 224,162 AMD Opteron processors with six cores each. The Tiahe-1A adopted a hybrid node design using two Xeon CPUs plus two AMD GPUs per each compute node. The GPU could be replaced by special floating-point accelerators. A homogeneous node design makes it easier to program and maintain the system.

Example 2.1 Modular Packaging of the IBM Blue Gene/L System

The Blue Gene/L is a supercomputer jointly developed by IBM and Lawrence Livermore National Laboratory. The system became operational in 2005 with a 136 Tflops performance at the No. 1 position in the

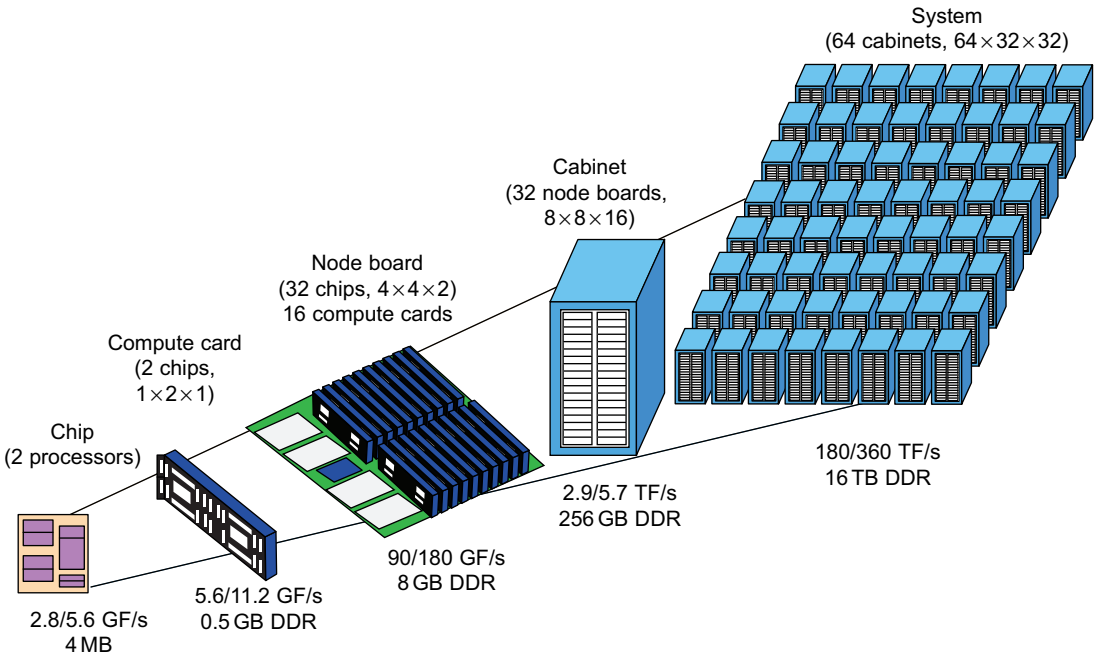


FIGURE 2.6

The IBM Blue Gene/L architecture built with modular components packaged hierarchically in five levels.

(*Courtesy of N. Adiga, et al., IBM Corp., 2005 [1]*)

Top-500 list—toped the Japanese Earth Simulator. The system was upgraded to score a 478 Tflops speed in 2007. By examining the architecture of the Blue Gene series, we reveal the modular construction of a scalable MPP system as shown in Figure 2.6. With modular packaging, the Blue Gene/L system is constructed hierarchically from processor chips to 64 physical racks. This system was built with a total of 65,536 nodes with two PowerPC 449 FP2 processors per node. The 64 racks are interconnected by a huge 3D 64 x 32 x 32 torus network.

In the lower-left corner, we see a dual-processor chip. Two chips are mounted on a computer card. Sixteen computer cards (32 chips or 64 processors) are mounted on a node board. A cabinet houses 32 node boards with an 8 x 8 x 16 torus interconnect. Finally, 64 cabinets (racks) form the total system at the upper-right corner. This packaging diagram corresponds to the 2005 configuration. Customers can order any size to meet their computational needs. The Blue Gene cluster was designed to achieve scalable performance, reliability through built-in testability, resilience by preserving locality of failures and checking mechanisms, and serviceability through partitioning and isolation of fault locations.

2.2.3 Cluster System Interconnects

2.2.3.1 High-Bandwidth Interconnects

Table 2.4 compares four families of high-bandwidth system interconnects. In 2007, Ethernet used a 1 Gbps link, while the fastest InfiniBand links ran at 30 Gbps. The Myrinet and Quadrics perform in between. The MPI latency represents the state of the art in long-distance message passing. All four technologies can implement any network topology, including crossbar switches, fat trees, and torus networks. The InfiniBand is the most expensive choice with the fastest link speed. The Ethernet is still the most cost-effective choice. We consider two example cluster interconnects over 1,024 nodes in Figure 2.7 and Figure 2.9. The popularity of five cluster interconnects is compared in Figure 2.8.

Table 2.4 Comparison of Four Cluster Interconnect Technologies Reported in 2007

Feature	Myrinet	Quadrics	InfiniBand	Ethernet
Available link speeds	1.28 Gbps (*M-XP*) 10 Gbps (*M-10G*)	2.8 Gbps (*QsNet*) 7.2 Gbps (*QsNetII*)	2.5 Gbps (*1X*) 10 Gbps (*4X*) 30 Gbps (*12X*)	1 Gbps
MPI latency	~3 us	~3 us	~4.5 us	~40 us
Network processor	Yes	Yes	Yes	No
Topologies	Any	Any	Any	Any
Network topology	Clos	Fat tree	Fat tree	Any
Routing	Source-based, cut-through	Source-based, cut-through	Destination-based	Destination-based
Flow control	Stop and go	Worm-hole	Credit-based	802.3x

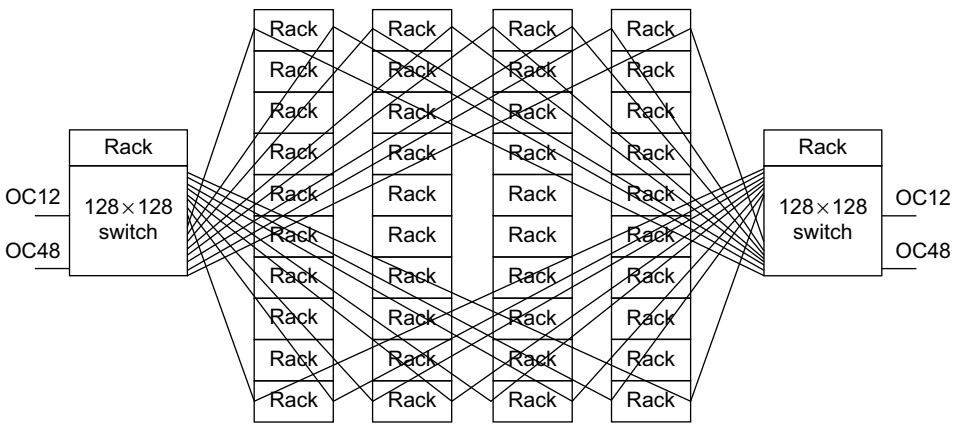


FIGURE 2.7

Google search engine cluster architecture.

(*Courtesy of Google, Inc. [6]*)

Example 2.2 Crossbar Switch in Google Search Engine Cluster

Google has many data centers using clusters of low-cost PC engines. These clusters are mainly used to support Google's web search business. Figure 2.7 shows a Google cluster interconnect of 40 racks of PC engines via two racks of 128 x 128 Ethernet switches. Each Ethernet switch can handle 128 one Gbps Ethernet links. A rack contains 80 PCs. This is an earlier cluster of 3,200 PCs. Google's search engine clusters are built with a lot more nodes. Today's server clusters from Google are installed in data centers with container trucks.

Two switches are used to enhance cluster availability. The cluster works fine even when one switch fails to provide the links among the PCs. The front ends of the switches are connected to the Internet via 2.4 Gbps OC 12 links. The 622 Mbps OC 12 links are connected to nearby data-center networks. In case of failure of the OC 48 links, the cluster is still connected to the outside world via the OC 12 links. Thus, the Google cluster eliminates all single points of failure.

2.2.3.2 Share of System Interconnects over Time

Figure 2.8 shows the distribution of large-scale system interconnects in the Top 500 systems from 2003 to 2008. Gigabit Ethernet is the most popular interconnect due to its low cost and market readiness. The InfiniBand network has been chosen in about 150 systems for its high-bandwidth performance. The Cray interconnect is designed for use in Cray systems only. The use of Myrinet and Quadrics networks had declined rapidly in the Top 500 list by 2008.

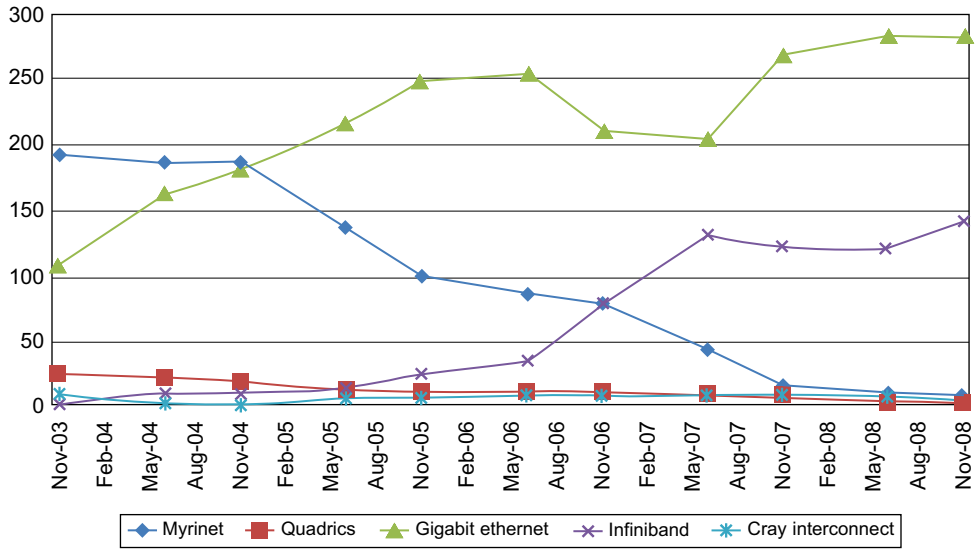


FIGURE 2.8

Distribution of high-bandwidth interconnects in the Top 500 systems from 2003 to 2008.

(*Courtesy of www.top500.org [25]*)

Example 2.3 Understanding the InfiniBand Architecture [8]

The InfiniBand has a switch-based point-to-point interconnect architecture. A large InfiniBand has a layered architecture. The interconnect supports the virtual interface architecture (VIA) for distributed messaging. The InfiniBand switches and links can make up any topology. Popular ones include crossbars, fat trees, and torus networks. Figure 2.9 shows the layered construction of an InfiniBand network. According to Table 2.5, the InfiniBand provides the highest speed links and the highest bandwidth in reported large-scale systems. However, InfiniBand networks cost the most among the four interconnect technologies.

Each end point can be a storage controller, a network interface card (NIC), or an interface to a host system. A host channel adapter (HCA) connected to the host processor through a standard peripheral component interconnect (PCI), PCI extended (PCI-X), or PCI express bus provides the host interface. Each HCA has more than one InfiniBand port. A target channel adapter (TCA) enables I/O devices to be loaded within the network. The TCA includes an I/O controller that is specific to its particular device's protocol such as SCSI, Fibre Channel, or Ethernet. This architecture can be easily implemented to build very large scale cluster interconnects that connect thousands or more hosts together. Supporting the InfiniBand in cluster applications can be found in [8].

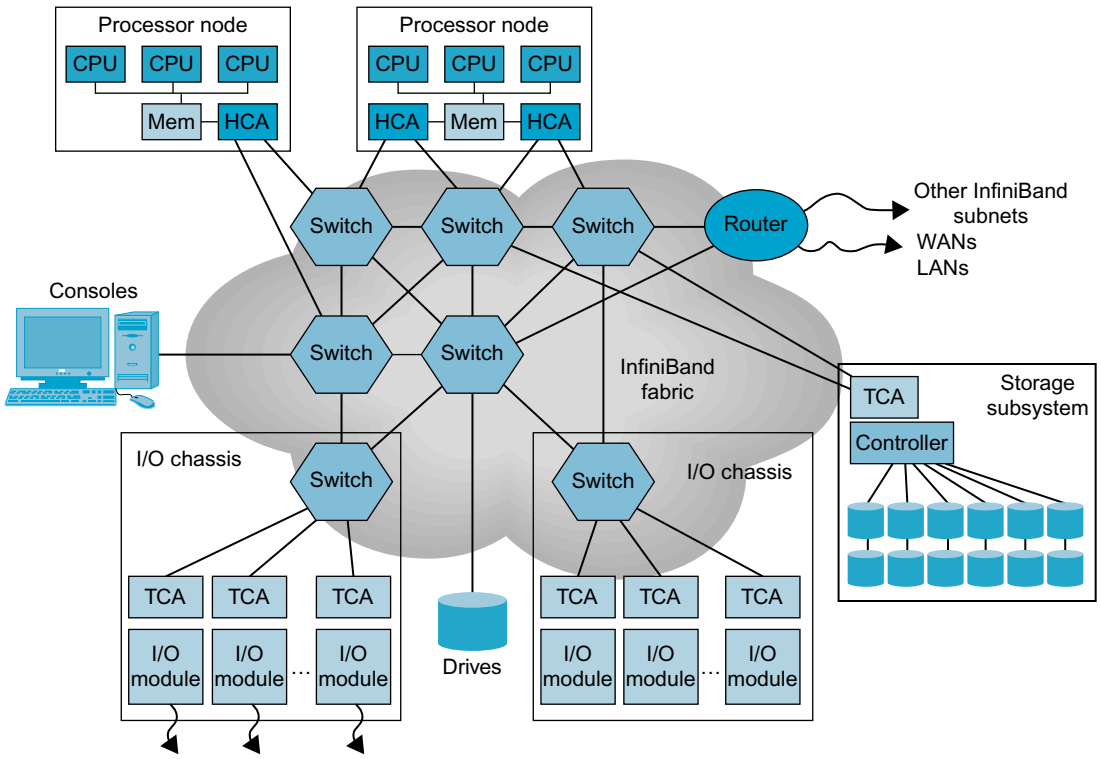


FIGURE 2.9

The InfiniBand system fabric built in a typical high-performance computer cluster.

(*Source: O. Celebioglu, et al, "Exploring InfiniBand as an HPC Cluster Interconnect", Dell Power Solutions, Oct.2004 © 2011 Dell Inc. All Rights Reserved*)

2.2.4 Hardware, Software, and Middleware Support

Realistically, SSI and HA features in a cluster are not obtained free of charge. They must be supported by hardware, software, middleware, or OS extensions. Any change in hardware design and OS extensions must be done by the manufacturer. The hardware and OS support could be cost-prohibitive to ordinary users. However, programming level is a big burden to cluster users. Therefore, the middleware support at the application level costs the least to implement. As an example, we show in Figure 2.10 the middleware, OS extensions, and hardware support needed to achieve HA in a typical Linux cluster system.

Close to the user application end, middleware packages are needed at the cluster management level: one for fault management to support *failover* and *failback*, to be discussed in Section 2.3.3. Another desired feature is to achieve HA using failure detection and recovery and packet switching. In the middle of Figure 2.10, we need to modify the Linux OS to support HA, and we need special drivers to support HA, I/O, and hardware devices. Toward the bottom, we need special hardware to support hot-swapped devices and provide router interfaces. We will discuss various supporting mechanisms in subsequent sections.

2.2.5 GPU Clusters for Massive Parallelism

Commodity GPUs are becoming high-performance accelerators for data-parallel computing. Modern GPU chips contain hundreds of processor cores per chip. Based on a 2010 report [19], each GPU

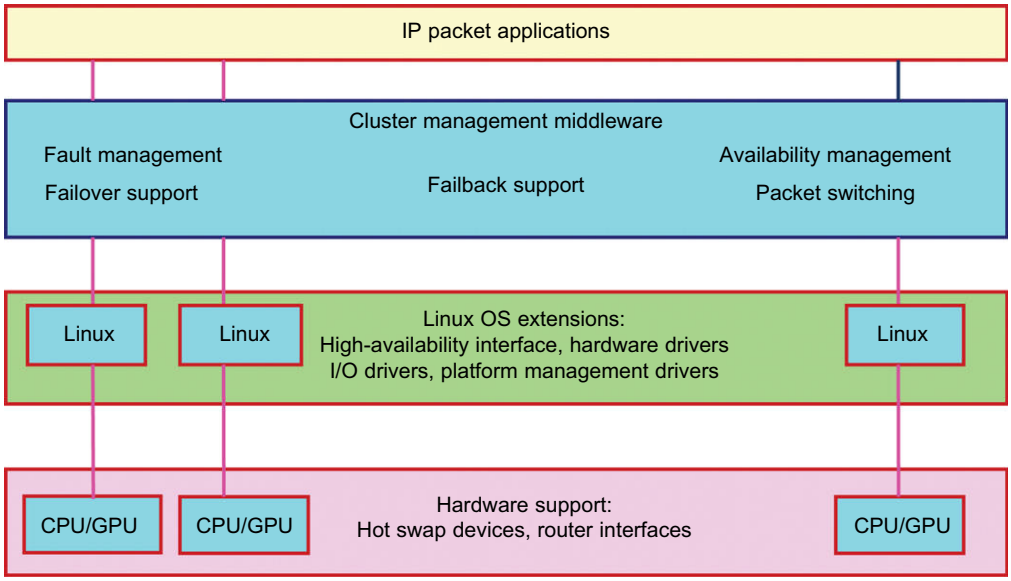


FIGURE 2.10

Middleware, Linux extensions, and hardware support for achieving massive parallelism and HA in a Linux cluster system built with CPUs and GPUs.

chip is capable of achieving up to 1 Tflops for single-precision (SP) arithmetic, and more than 80 Gflops for double-precision (DP) calculations. Recent HPC-optimized GPUs contain up to 4 GB of on-board memory, and are capable of sustaining memory bandwidths exceeding 100 GB/second. GPU clusters are built with a large number of GPU chips. GPU clusters have already demonstrated their capability to achieve Pflops performance in some of the Top 500 systems. Most GPU clusters are structured with homogeneous GPUs of the same hardware class, make, and model. The software used in a GPU cluster includes the OS, GPU drivers, and clustering API such as an MPI.

The high performance of a GPU cluster is attributed mainly to its massively parallel multicore architecture, high throughput in multithreaded floating-point arithmetic, and significantly reduced time in massive data movement using large on-chip cache memory. In other words, GPU clusters already are more cost-effective than traditional CPU clusters. GPU clusters result in not only a quantum jump in speed performance, but also significantly reduced space, power, and cooling demands. A GPU cluster can operate with a reduced number of operating system images, compared with CPU-based clusters. These reductions in power, environment, and management complexity make GPU clusters very attractive for use in future HPC applications.

2.2.5.1 The Echelon GPU Chip Design

Figure 2.11 shows the architecture of a future GPU accelerator that was suggested for use in building a NVIDIA Echelon GPU cluster for Exascale computing. This Echelong project led by Bill Dally at NVIDIA is partially funded by DARPA under the Ubiquitous High-Performance Computing (UHPC) program. This GPU design incorporates 1024 stream cores and 8 latency-optimized CPU-like cores (called latency processor) on a single chip. Eight stream cores form a *stream multiprocessor* (SM) and there are 128 SMs in the Echelon GPU chip.

Each SM is designed with 8 processor cores to yield a 160 Gflops peak speed. With 128 SMs, the chip has a peak speed of 20.48 Tflops. These nodes are interconnected by a NoC (network on chip) to 1,024 SRAM banks (L2 caches). Each cache band has a 256 KB capacity. The MCs (*memory controllers*) are used to connect to off-chip DRAMs and the NI (*network interface*) is to scale the size of the GPU cluster hierarchically, as shown in Figure 2.14. At the time of this writing, the Echelon is only a research project. With permission from Bill Dally, we present the design for

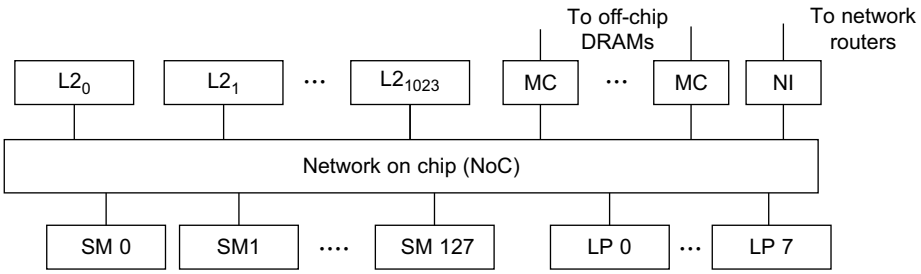


FIGURE 2.11

The proposed GPU chip design for 20 Tflops performance and 1.6 TB/s memory bandwidth in the Echelon system.
(*Courtesy of Bill Dally, Reprinted with Permission [10]*)

academic interest to illustrate how one can explore the many-core GPU technology to achieve Exascale computing in the future PU technology to achieve Exascale computing in the future.

2.2.5.2 GPU Cluster Components

A GPU cluster is often built as a heterogeneous system consisting of three major components: the CPU host nodes, the GPU nodes and the cluster interconnect between them. The GPU nodes are formed with general-purpose GPUs, known as GPGPUs, to carry out numerical calculations. The host node controls program execution. The cluster interconnect handles inter-node communications. To guarantee the performance, multiple GPUs must be fully supplied with data streams over high-bandwidth network and memory. Host memory should be optimized to match with the on-chip cache bandwidths on the GPUs. Figure 2.12 shows the proposed Echelon GPU clusters using the GPU chips shown in Figure 2.13 as building blocks interconnected by a hierarchically constructed network.

2.2.5.3 Echelon GPU Cluster Architecture

The Echelon system architecture is shown in Figure 2.11, hierarchically. The entire Echelon system is built with *N* cabinets, labeled C0, C1, ... , C*N*. Each cabinet is built with 16 compute module labeled as M0, M1, ... , M15. Each compute module is built with 8 GPU nodes labeled as N0, N1, ... , N7. Each GPU node is the innermost block labeled as PC in Figure 2.12 (also detailed in Figure 2.11).

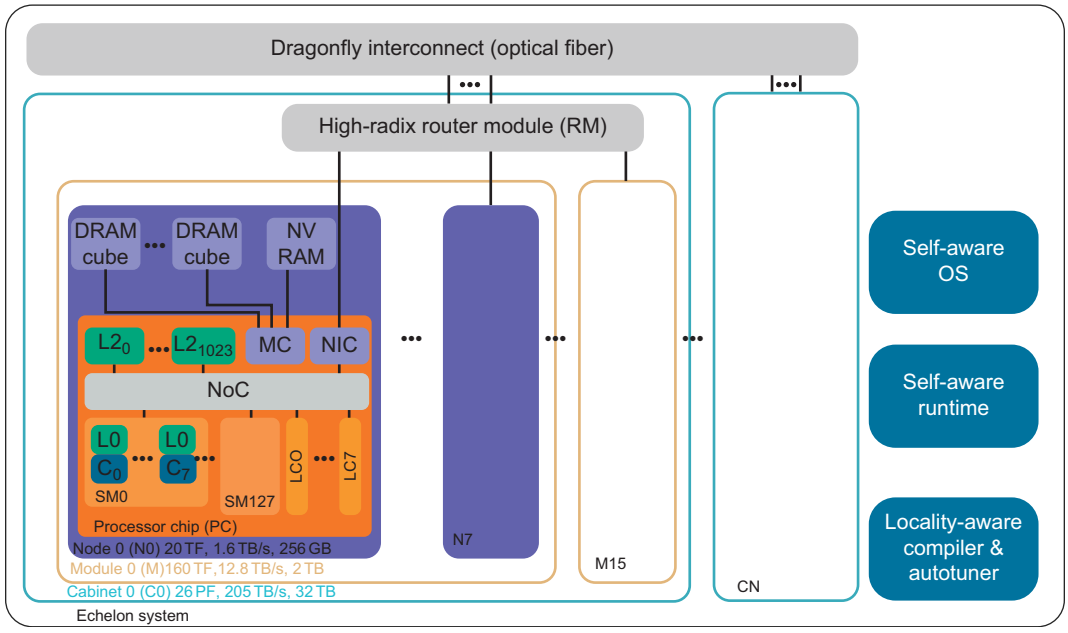


FIGURE 2.12

The architecture of NVIDIA Echelon system built with a hierarchical network of GPUs that can deliver 2.6 Pflops per cabinet, and takes at least N = 400 cabinets to achieve the desired Eflops performance.

(*Courtesy of Bill Dally, Reprinted with Permission [10]*)

Each compute module features a performance of 160 Tflops and 12.8 TB/s over 2 TB of memory. Thus, a single cabinet can house 128 GPU nodes or 16,000 processor cores. Each cabinet has the potential to deliver 2.6 Pflops over 32 TB memory and 205 TB/s bandwidth. The *N* cabinets are interconnected by a Dragonfly network with optical fiber.

To achieve Eflops performance, we need to use at least $N = 400$ cabinets. In total, an Exascale system needs 327,680 processor cores in 400 cabinets. The Echelon system is supported by a self-aware OS and runtime system. The Echelon system is also designed to preserve locality with the support of compiler and autotuner. At present, NVIDIA Fermi (GF110) chip has 512 stream processors. Thus the Echelon design is about 25 times faster. It is highly likely that the Echelon will employ post-Maxwell NVIDIA GPU planned to appear in 2013 ~ 2014 time frame.

2.2.5.4 CUDA Parallel Programming

CUDA (Compute Unified Device Architecture) offers a parallel computing architecture developed by NVIDIA. CUDA is the computing engine in NVIDIA GPUs. This software is accessible to developers through standard programming languages. Programmers use C for CUDA C with NVIDIA extensions and certain restrictions. This CUDA C is compiled through a PathScale Open64 C compiler for parallel execution on a large number of GPU cores. Example 2.4 shows the advantage of using CUDA C in parallel processing.

Example 2.4 Parallel SAXPY Execution Using CUDA C Code on GPUs

SAXPY is a kernel operation frequently performed in matrix multiplication. It essentially performs repeated *multiply and add* operations to generate the dot product of two long vectors. The following `saxpy_serial` routine is written in standard C code. This code is only suitable for sequential execution on a single processor core.

```
Void saxpy_serial (int n, float a, float*x, float *
  { for (int i = 0; i < n; ++i), y[i] = a*x[i] + y[i] }
    // Invoke the serial SAXPY kernel
  saxpy_serial (n, 2.0, x, y);
```

The following `saxpy_parallel` routine is written in CUDA C code for parallel execution by 256 threads/block on many processing cores on the GPU chip. Note that `n` blocks are handled by `n` processing cores, where `n` could be on the order of hundreds of blocks.

```
_global__void saxpy_parallel (int n, float a, float*x, float *y)
  { Int i = blockIndex.x*blockDim.x + threadIndex.x; if (i < n) y [i] = a*x[i] + y[i] }
 // Invoke the parallel SAXPY kernel with 256 threads/block int nblocks = (n + 255)/256;
  saxpy_parallel <<< nblocks, 256 >>> (n, 2.0, x, y);
```

This is a good example of using CUDA C to exploit massive parallelism on a cluster of multicore and multithreaded processors using the CODA GPGPUs as building blocks.

2.2.5.5 CUDA Programming Interfaces

CUDA architecture shares a range of computational interfaces with two competitors: the *Khronos Group*'s *Open Computing Language* and Microsoft's *DirectCompute*. Third-party wrappers are also

available for using Python, Perl, FORTRAN, Java, Ruby, Lua, MATLAB, and IDL. CUDA has been used to accelerate nongraphical applications in computational biology, cryptography, and other fields by an order of magnitude or more. A good example is the BOINC distributed computing client. CUDA provides both a low-level API and a higher-level API. CUDA works with all NVIDIA GPUs from the G8X series onward, including the GeForce, Quadro, and Tesla lines. NVIDIA states that programs developed for the GeForce 8 series will also work without modification on all future NVIDIA video cards due to binary compatibility.

2.2.5.6 Trends in CUDA Usage

Tesla and Fermi are two generations of CUDA architecture released by NVIDIA in 2007 and 2010, respectively. The CUDA version 3.2 is used for using a single GPU module in 2010. A newer CUDA version 4.0 will allow multiple GPUs to address use an unified virtual address space of shared memory. The next NVIDIA GPUs will be Kepler-designed to support C++. The Fermi has eight times the peak double-precision floating-point performance of the Tesla GPU (5.8 Gflops/W versus 0.7 Gflops/W). Currently, the Fermi has up to 512 CUDA cores on 3 billion transistors.

Future applications of the CUDA GPUs and the Echelon system may include the following:

- The search for extraterrestrial intelligence (SETI@Home)
- Distributed calculations to predict the native conformation of proteins
- Medical analysis simulations based on CT and MRI scan images
- Physical simulations in fluid dynamics and environment statistics
- Accelerated 3D graphics, cryptography, compression, and interconversion of video file formats
- Building the *single-chip cloud computer* (SCC) through virtualization in many-core architecture.

2.3 DESIGN PRINCIPLES OF COMPUTER CLUSTERS

Clusters should be designed for scalability and availability. In this section, we will cover the design principles of SSI, HA, fault tolerance, and rollback recovery in general-purpose computers and clusters of cooperative computers.

2.3.1 Single-System Image Features

SSI does not mean a single copy of an operating system image residing in memory, as in an SMP or a workstation. Rather, it means the *illusion* of a single system, single control, symmetry, and transparency as characterized in the following list:

- **Single system** The entire cluster is viewed by users as one system that has multiple processors. The user could say, "Execute my application using five processors." This is different from a distributed system.
- **Single control** Logically, an end user or system user utilizes services from one place with a single interface. For instance, a user submits batch jobs to one set of queues; a system administrator configures all the hardware and software components of the cluster from one control point.

- **Symmetry** A user can use a cluster service from any node. In other words, all cluster services and functionalities are symmetric to all nodes and all users, except those protected by access rights.
- **Location-transparent** The user is not aware of the where abouts of the physical device that eventually provides a service. For instance, the user can use a tape drive attached to any cluster node as though it were physically attached to the local node.

The main motivation to have SSI is that it allows a cluster to be used, controlled, and maintained as a familiar workstation is. The word "single" in "single-system image" is sometimes synonymous with "global" or "central." For instance, a global file system means a single file hierarchy, which a user can access from any node. A single point of control allows an operator to monitor and configure the cluster system. Although there is an illusion of a single system, a cluster service or functionality is often realized in a distributed manner through the cooperation of multiple components. A main requirement (and advantage) of SSI techniques is that they provide both the performance benefits of distributed implementation and the usability benefits of a single image.

From the viewpoint of a process *P*, cluster nodes can be classified into three types. The *home node* of a process *P* is the node where *P* resided when it was created. The *local node* of a process *P* is the node where *P* currently resides. All other nodes are *remote nodes* to *P*. Cluster nodes can be configured to suit different needs. A *host node* serves user logins through Telnet, rlogin, or even FTP and HTTP. A *compute node* is one that performs computational jobs. An *I/O node* is one that serves file I/O requests. If a cluster has large shared disks and tape units, they are normally physically attached to I/O nodes.

There is one home node for each process, which is fixed throughout the life of the process. At any time, there is only one local node, which may or may not be the host node. The local node and remote nodes of a process may change when the process migrates. A node can be configured to provide multiple functionalities. For instance, a node can be designated as a host, an I/O node, and a compute node at the same time. The illusion of an SSI can be obtained at several layers, three of which are discussed in the following list. Note that these layers may overlap with one another.

- **Application software layer** Two examples are parallel web servers and various parallel databases. The user sees an SSI through the application and is not even aware that he is using a cluster. This approach demands the modification of workstation or SMP applications for clusters.
- **Hardware or kernel layer** Ideally, SSI should be provided by the operating system or by the hardware. Unfortunately, this is not a reality yet. Furthermore, it is extremely difficult to provide an SSI over heterogeneous clusters. With most hardware architectures and operating systems being proprietary, only the manufacturer can use this approach.
- **Middleware layer** The most viable approach is to construct an SSI layer just above the OS kernel. This approach is promising because it is platform-independent and does not require application modification. Many cluster job management systems have already adopted this approach.

Each computer in a cluster has its own operating system image. Thus, a cluster may display multiple system images due to the stand-alone operations of all participating node computers. Determining how to merge the multiple system images in a cluster is as difficult as regulating many individual personalities in a community to a single personality. With different degrees of resource sharing, multiple systems could be integrated to achieve SSI at various operational levels.

2.3.1.1 Single Entry Point

Single-system image (SSI) is a very rich concept, consisting of single entry point, single file hierarchy, single I/O space, single networking scheme, single control point, single job management system, single memory space, and single process space. The single entry point enables users to log in (e.g., through Telnet, rlogin, or HTTP) to a cluster as one virtual host, although the cluster may have multiple physical host nodes to serve the login sessions. The system transparently distributes the user's login and connection requests to different physical hosts to balance the load. Clusters could substitute for mainframes and supercomputers. Also, in an Internet cluster server, thousands of HTTP or FTP requests may come simultaneously. Establishing a single entry point with multiple hosts is not a trivial matter. Many issues must be resolved. The following is just a partial list:

- **Home directory** Where do you put the user's home directory?
- **Authentication** How do you authenticate user logins?
- **Multiple connections** What if the same user opens several sessions to the same user account?
- **Host failure** How do you deal with the failure of one or more hosts?

Example 2.5 Realizing a Single Entry Point in a Cluster of Computers

Figure 2.13 illustrates how to realize a single entry point. Four nodes of a cluster are used as host nodes to receive users' login requests. Although only one user is shown, thousands of users can connect to the cluster in the same fashion. When a user logs into the cluster, he issues a standard UNIX command such as *telnet* cluster.cs.hku.hk, using the symbolic name of the cluster system.

The DNS translates the symbolic name and returns the IP address 159.226.41.150 of the least-loaded node, which happens to be node Host1. The user then logs in using this IP address. The DNS periodically receives load information from the host nodes to make load-balancing translation decisions. In the ideal case, if 200 users simultaneously log in, the login sessions are evenly distributed among our hosts with 50 users each. This allows a single host to be four times more powerful.

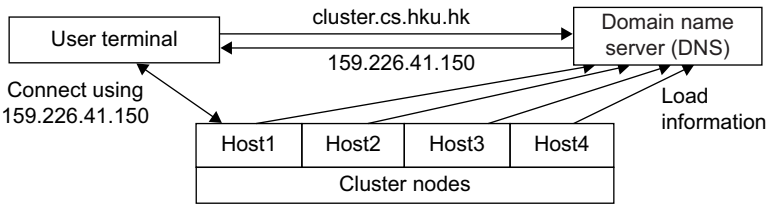


FIGURE 2.13

Realizing a single entry point using a load-balancing domain name system (DNS).

(*Courtesy of Hwang and Xu [14]*)

2.3.1.2 Single File Hierarchy

We use the term "single file hierarchy" in this book to mean the illusion of a single, huge file system image that transparently integrates local and global disks and other file devices (e.g., tapes). In other words, all files a user needs are stored in some subdirectories of the root directory /, and they can be accessed through ordinary UNIX calls such as *open*, *read*, and so on. This should not be confused with the fact that multiple file systems can exist in a workstation as subdirectories of the root directory.

The functionalities of a single file hierarchy have already been partially provided by existing distributed file systems such as *Network File System (NFS)* and *Andrew File System (AFS)*. From the viewpoint of any process, files can reside on three types of locations in a cluster, as shown in Figure 2.14.

Local storage is the disk on the local node of a process. The disks on remote nodes are *remote storage*. A *stable storage* requires two aspects: It is *persistent*, which means data, once written to the stable storage, will stay there for a sufficiently long time (e.g., a week), even after the cluster shuts down; and it is fault-tolerant to some degree, by using redundancy and periodic backup to tapes.

Figure 2.14 uses stable storage. Files in stable storage are called *global files*, those in local storage *local files*, and those in remote storage *remote files*. Stable storage could be implemented as one centralized, large RAID disk. But it could also be distributed using local disks of cluster nodes. The first approach uses a large disk, which is a single point of failure and a potential performance bottleneck. The latter approach is more difficult to implement, but it is potentially more economical, more efficient, and more available. On many cluster systems, it is customary for the system to make visible to the user processes the following directories in a single file hierarchy: the usual *system directories* as in a traditional UNIX workstation, such as /usr and /usr/local; and the user's *home directory* ~/ that has a small disk quota (1–20 MB). The user stores his code files and other files here. But large data files must be stored elsewhere.

- A *global directory* is shared by all users and all processes. This directory has a large disk space of multiple gigabytes. Users can store their large data files here.
- On a cluster system, a process can access a special directory on the local disk. This directory has medium capacity and is faster to access than the global directory.

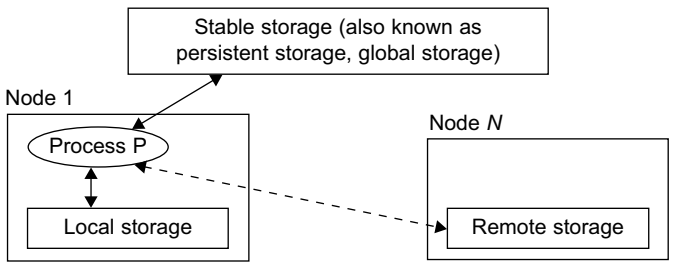


FIGURE 2.14

Three types of storage in a single file hierarchy. Solid lines show what process P can access and the dashed line shows what P may be able to access.

(*Courtesy of Hwang and Xu [14]*)

2.3.1.3 Visibility of Files

The term "visibility" here means a process can use traditional UNIX system or library calls such as *fopen*, *fread*, and *fwrite* to access files. Note that there are multiple local scratch directories in a cluster. The local scratch directories in remote nodes are not in the single file hierarchy, and are not directly visible to the process. A user process can still access them with commands such as *rcp* or some special library functions, by specifying both the node name and the filename.

The name "scratch" indicates that the storage is meant to act as a scratch pad for temporary information storage. Information in the local scratch space could be lost once the user logs out. Files in the global scratch space will normally persist even after the user logs out, but will be deleted by the system if not accessed in a predetermined time period. This is to free disk space for other users. The length of the period can be set by the system administrator, and usually ranges from one day to several weeks. Some systems back up the global scratch space to tapes periodically or before deleting any files.

2.3.1.4 Support of Single-File Hierarchy

It is desired that a single file hierarchy have the SSI properties discussed, which are reiterated for file systems as follows:

- **Single system** There is just one file hierarchy from the user's viewpoint.
- **Symmetry** A user can access the global storage (e.g., */scratch*) using a cluster service from any node. In other words, all file services and functionalities are symmetric to all nodes and all users, except those protected by access rights.
- **Location-transparent** The user is not aware of the whereabouts of the physical device that eventually provides a service. For instance, the user can use a RAID attached to any cluster node as though it were physically attached to the local node. There may be some performance differences, though.

A cluster file system should maintain *UNIX semantics*: Every file operation (*fopen*, *fread*, *fwrite*, *fclose*, etc.) is a transaction. When an *fread* accesses a file after an *fwrite* modifies the same file, the *fread* should get the updated value. However, existing distributed file systems do not completely follow UNIX semantics. Some of them update a file only at close or flush. A number of alternatives have been suggested to organize the global storage in a cluster. One extreme is to use a single file server that hosts a big RAID. This solution is simple and can be easily implemented with current software (e.g., NFS). But the file server becomes both a performance bottleneck and a single point of failure. Another extreme is to utilize the local disks in all nodes to form global storage. This could solve the performance and availability problems of a single file server.

2.3.1.5 Single I/O, Networking, and Memory Space

To achieve SSI, we desire a single control point, a single address space, a single job management system, a single user interface, and a single process control, as depicted in Figure 2.17. In this example, each node has exactly one network connection. Two of the four nodes each have two I/O devices attached.

Single Networking: A properly designed cluster should behave as one system (the shaded area). In other words, it is like a big workstation with four network connections and four I/O devices

attached. Any process on any node can use any network and I/O device as though it were attached to the local node. Single networking means any node can access any network connection.

Single Point of Control: The system administrator should be able to configure, monitor, test, and control the entire cluster and each individual node from a single point. Many clusters help with this through a system console that is connected to all nodes of the cluster. The system console is normally connected to an external LAN (not shown in Figure 2.15) so that the administrator can log in remotely to the system console from anywhere in the LAN to perform administration work.

Note that single point of control does not mean all system administration work should be carried out solely by the system console. In reality, many administrative functions are distributed across the cluster. It means that controlling a cluster should be no more difficult than administering an SMP or a mainframe. It implies that administration-related system information (such as various configuration files) should be kept in one logical place. The administrator monitors the cluster with one graphics tool, which shows the entire picture of the cluster, and the administrator can zoom in and out at will.

Single point of control (or *single point of management*) is one of the most challenging issues in constructing a cluster system. Techniques from distributed and networked system management can be transferred to clusters. Several de facto standards have already been developed for network management. An example is *Simple Network Management Protocol (SNMP)*. It demands an efficient cluster management package that integrates with the availability support system, the file system, and the job management system.

Single Memory Space: *Single memory space* gives users the illusion of a big, centralized main memory, which in reality may be a set of distributed local memory spaces. PVPs, SMPs, and DSMs have an edge over MPPs and clusters in this respect, because they allow a program to utilize all global or local memory space. A good way to test if a cluster has a single memory space is to run a *sequential* program that needs a memory space larger than any single node can provide.

Suppose each node in Figure 2.15 has 2 GB of memory available to users. An ideal single memory image would allow the cluster to execute a sequential program that needs 8 GB of memory. This would enable a cluster to operate like an SMP system. Several approaches have

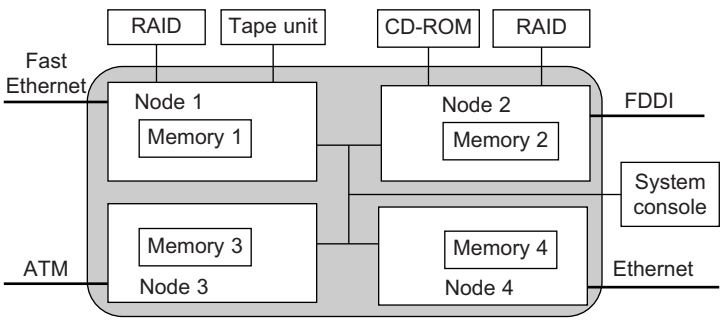


FIGURE 2.15

A cluster with single networking, single I/O space, single memory, and single point of control.

(*Courtesy of Hwang and Xu [14]*)

been attempted to achieve a single memory space on clusters. Another approach is to let the compiler distribute the data structures of an application across multiple nodes. It is still a challenging task to develop a single memory scheme that is efficient, platform-independent, and able to support sequential binary codes.

Single I/O Address Space: Assume the cluster is used as a web server. The web information database is distributed between the two RAIDs. An HTTP daemon is started on each node to handle web requests, which come from all four network connections. A single I/O space implies that any node can access the two RAIDs. Suppose most requests come from the ATM network. It would be beneficial if the functions of the HTTP on node 3 could be distributed to all four nodes. The following example shows a distributed RAID-x architecture for I/O-centric cluster computing [9].

Example 2.6 Single I/O Space over Distributed RAID for I/O-Centric Clusters

A distributed disk array architecture was proposed by Hwang, et al. [9] for establishing a single I/O space in I/O-centric cluster applications. Figure 2.16 shows the architecture for a four-node Linux PC cluster, in which three disks are attached to the SCSI bus of each host node. All 12 disks form an integrated RAID-x with a single address space. In other words, all PCs can access both local and remote disks. The addressing scheme for all disk blocks is interleaved horizontally. Orthogonal stripping and mirroring make it possible to have a RAID-1 equivalent capability in the system.

The shaded blocks are images of the blank blocks. A disk block and its image will be mapped on different physical disks in an orthogonal manner. For example, the block B_0 is located on disk D0. The image block M_0 of block B_0 is located on disk D3. The four disks D0, D1, D2, and D3 are attached to four servers, and thus can be accessed in parallel. Any single disk failure will not lose the data block, because its image is available in recovery. All disk blocks are labeled to show image mapping. Benchmark experiments show that this RAID-x is scalable and can restore data after any single disk failure. The distributed RAID-x has improved aggregate I/O bandwidth in both parallel read and write operations over all physical disks in the cluster.

2.3.1.6 Other Desired SSI Features

The ultimate goal of SSI is for a cluster to be as easy to use as a desktop computer. Here are additional types of SSI, which are present in SMP servers:

- **Single job management system** All cluster jobs can be submitted from any node to a single job management system.
- **Single user interface** The users use the cluster through a single graphical interface. Such an interface is available for workstations and PCs. A good direction to take in developing a cluster GUI is to utilize web technology.
- **Single process space** All user processes created on various nodes form a single process space and share a uniform process identification scheme. A process on any node can create (e.g., through a UNIX fork) or communicate with (e.g., through signals, pipes, etc.) processes on remote nodes.

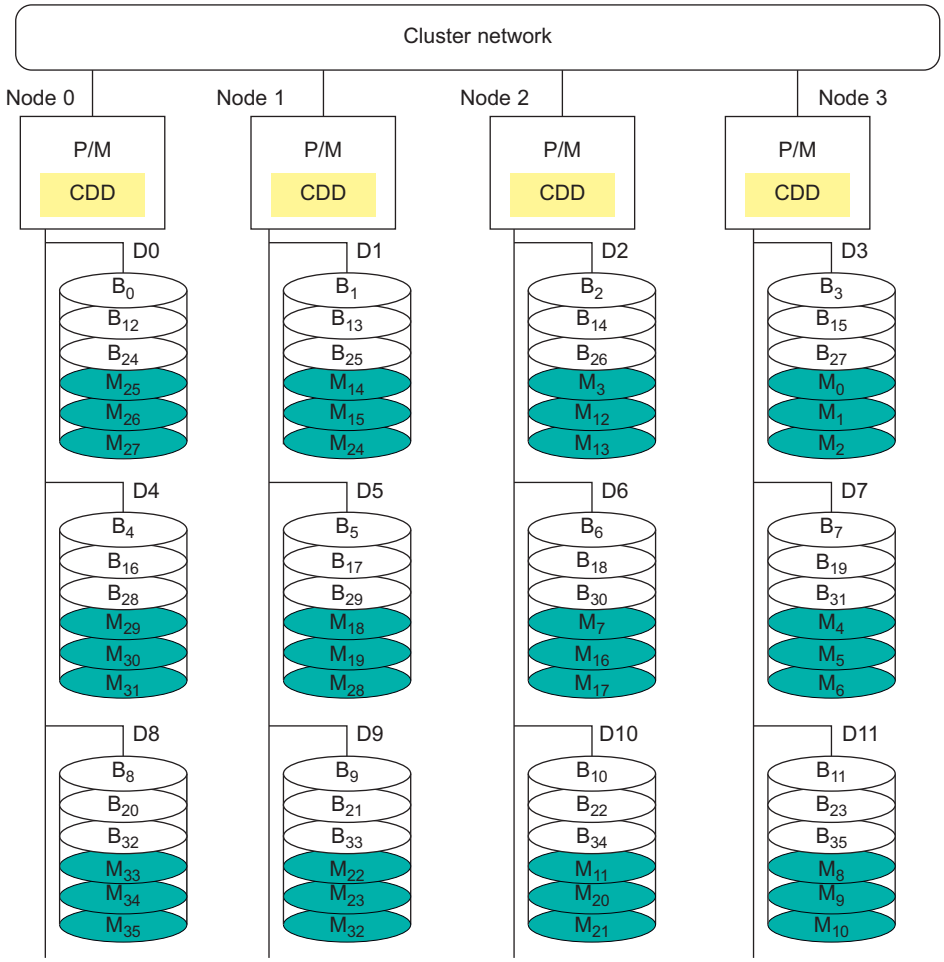


FIGURE 2.16

Distributed RAID architecture with a single I/O space over 12 distributed disks attached to 4 host computers in the cluster (Di stands for Disk i, B_j for disk block j, M_j an image for blocks B_j, P/M for processor/memory node, and CDD for cooperative disk driver.)

(*Courtesy of Hwang, Jin, and Ho [13]*)

- **Middleware support for SSI clustering** As shown in Figure 2.17, various SSI features are supported by middleware developed at three cluster application levels:
- **Management level** This level handles user applications and provides a job management system such as GLUnix, MOSIX, *Load Sharing Facility (LSF)*, or Codine.
- **Programming level** This level provides single file hierarchy (NFS, xFS, AFS, Proxy) and distributed shared memory (TreadMark, Wind Tunnel).

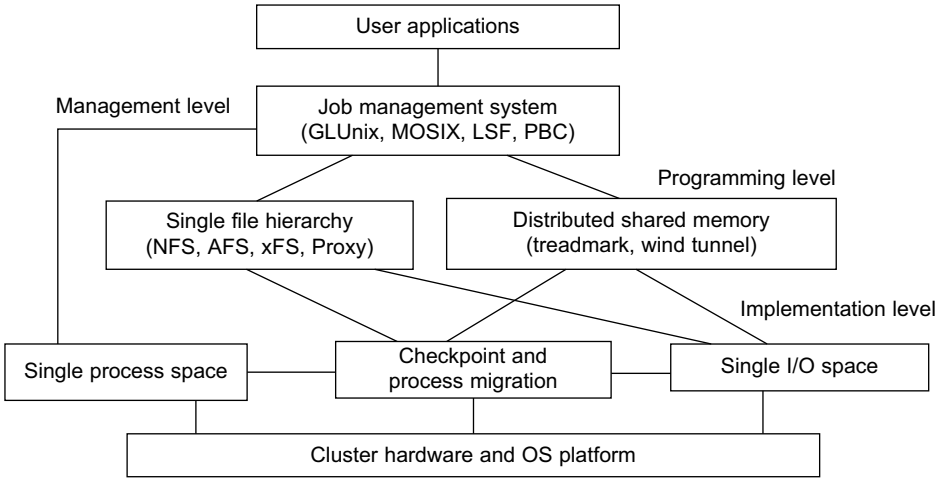


FIGURE 2.17

Relationship among clustering middleware at the job management, programming, and implementation levels.

(*Courtesy of K. Hwang, H. Jin, C.L. Wang and Z. Xu [16]*)

- **Implementation level** This level supports a single process space, checkpointing, process migration, and a single I/O space. These features must interface with the cluster hardware and OS platform. The distributed disk array, RAID-x, in Example 2.6 implements a single I/O space.

2.3.2 High Availability through Redundancy

When designing robust, highly available systems three terms are often used together: *reliability*, *availability*, and *serviceability (RAS)*. Availability is the most interesting measure since it combines the concepts of reliability and serviceability as defined here:

- *Reliability* measures how long a system can operate without a breakdown.
- *Availability* indicates the percentage of time that a system is available to the user, that is, the percentage of system uptime.
- *Serviceability* refers to how easy it is to service the system, including hardware and software maintenance, repair, upgrades, and so on.

The demand for RAS is driven by practical market needs. A recent Find/SVP survey found the following figures among Fortune 1000 companies: An average computer is down nine times per year with an average downtime of four hours. The average loss of revenue per hour of downtime is $82,500. With such a hefty penalty for downtime, many companies are striving for systems that offer *24/365 availability*, meaning the system is available 24 hours per day, 365 days per year.

2.3.2.1 Availability and Failure Rate

As Figure 2.18 shows, a computer system operates normally for a period of time before it fails. The failed system is then repaired, and the system returns to normal operation. This operate-repair cycle

then repeats. A system's reliability is measured by the *mean time to failure* (MTTF), which is the average time of normal operation before the system (or a component of the system) fails. The metric for serviceability is the *mean time to repair* (MTTR), which is the average time it takes to repair the system and restore it to working condition after it fails. The *availability* of a system is defined by:

$$Availability = MTTF/(MTTF + MTTR) \tag{2.1}$$

2.3.2.2 Planned versus Unplanned Failure

When studying RAS, we call any event that prevents the system from normal operation a *failure*. This includes:

- **Unplanned failures** The system breaks, due to an operating system crash, a hardware failure, a network disconnection, human operation errors, a power outage, and so on. All these are simply called failures. The system must be repaired to correct the failure.
- **Planned shutdowns** The system is not broken, but is periodically taken off normal operation for upgrades, reconfiguration, and maintenance. A system may also be shut down for weekends or holidays. The MTTR in Figure 2.18 for this type of failure is the planned downtime.

Table 2.5 shows the availability values of several representative systems. For instance, a conventional workstation has an availability of 99 percent, meaning it is up and running 99 percent of the time or it has a downtime of 3.6 days per year. An optimistic definition of availability does not consider planned downtime, which may be significant. For instance, many supercomputer installations have a planned downtime of several hours per week, while a telephone system cannot tolerate a downtime of a few minutes per year.

2.3.2.3 Transient versus Permanent Failures

A lot of failures are *transient* in that they occur temporarily and then disappear. They can be dealt with without replacing any components. A standard approach is to roll back the system to a known

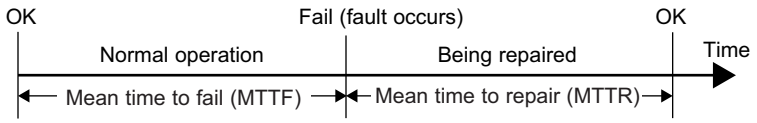


FIGURE 2.18

The operate-repair cycle of a computer system.

Table 2.5 Availability of Computer System Types

System Type	Availability (%)	Downtime in a Year
Conventional workstation	99	3.6 days
HA system	99.9	8.5 hours
Fault-resilient system	99.99	1 hour
Fault-tolerant system	99.999	5 minutes

state and start over. For instance, we all have rebooted our PC to take care of transient failures such as a frozen keyboard or window. *Permanent failures* cannot be corrected by rebooting. Some hardware or software component must be repaired or replaced. For instance, rebooting will not work if the system hard disk is broken.

2.3.2.4 Partial versus Total Failures

A failure that renders the entire system unusable is called a *total failure*. A failure that only affects part of the system is called a *partial failure* if the system is still usable, even at a reduced capacity. A key approach to enhancing availability is to make as many failures as possible partial failures, by systematically removing *single points of failure*, which are hardware or software components whose failure will bring down the entire system.

Example 2.7 Single Points of Failure in an SMP and in Clusters of Computers

In an SMP (Figure 2.19(a)), the shared memory, the OS image, and the memory bus are all single points of failure. On the other hand, the processors are not forming a single point of failure. In a cluster of workstations (Figure 2.19(b)), interconnected by Ethernet, there are multiple OS images, each residing in a workstation. This avoids the single point of failure caused by the OS as in the SMP case. However, the Ethernet network now becomes a single point of failure, which is eliminated in Figure 2.21(c), where a high-speed network is added to provide two paths for communication.

When a node fails in the clusters in Figure 2.19(b) and Figure 2.19(c), not only will the node applications all fail, but also all node data cannot be used until the node is repaired. The shared disk cluster in Figure 2.19(d) provides a remedy. The system stores persistent data on the shared disk, and periodically *checkpoints* to save intermediate results. When one WS node fails, the data will not be lost in this shared-disk cluster.

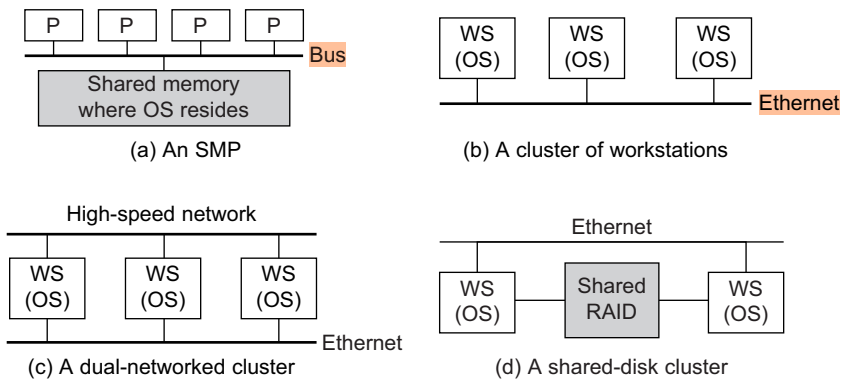


FIGURE 2.19

Single points of failure (SPF) in an SMP and in three clusters, where greater redundancy eliminates more SPFs in systems from (a) to (d).

(*Courtesy of Hwang and Xu [14]*)

2.3.2.5 Redundancy Techniques

Consider the cluster in Figure 2.19(d). Assume only the nodes can fail. The rest of the system (e.g., interconnect and the shared RAID disk) is 100 percent available. Also assume that when a node fails, its workload is switched over to the other node in zero time. We ask, what is the availability of the cluster if planned downtime is ignored? What is the availability if the cluster needs one hour/week for maintenance? What is the availability if it is shut down one hour/week, one node at a time?

According to Table 2.4, a workstation is available 99 percent of the time. The time both nodes are down is only 0.01 percent. Thus, the availability is 99.99 percent. It is now a fault-resilient system, with only one hour of downtime per year. The planned downtime is 52 hours per year, that is, $52 / (365 \times 24) = 0.0059$. The total downtime is now 0.59 percent + 0.01 percent = 0.6 percent. The availability of the cluster becomes 99.4 percent. Suppose we ignore the unlikely situation in which the other node fails while one node is maintained. Then the availability is 99.99 percent.

There are basically two ways to increase the availability of a system: increasing MTTF or reducing MTTR. Increasing MTTF amounts to increasing the reliability of the system. The computer industry has strived to make reliable systems, and today's workstations have MTTFs in the range of hundreds to thousands of hours. However, to further improve MTTF is very difficult and costly. Clusters offer an HA solution based on reducing the MTTR of the system. A multinode cluster has a lower MTTF (thus lower reliability) than a workstation. However, the failures are taken care of quickly to deliver higher availability. We consider several redundancy techniques used in cluster design.

2.3.2.6 Isolated Redundancy

A key technique to improve availability in any system is to use redundant components. When a component (the *primary* component) fails, the service it provided is taken over by another component (the *backup* component). Furthermore, the primary and the backup components should be *isolated* from each other, meaning they should not be subject to the same cause of failure. Clusters provide HA with redundancy in power supplies, fans, processors, memories, disks, I/O devices, networks, operating system images, and so on. In a carefully designed cluster, redundancy is also isolated. Isolated redundancy provides several benefits:

- First, a component designed with isolated redundancy is not a single point of failure, and the failure of that component will not cause a total system failure.
- Second, the failed component can be repaired while the rest of the system is still working.
- Third, the primary and the backup components can mutually test and debug each other.

The IBM SP2 communication subsystem is a good example of isolated-redundancy design. All nodes are connected by two networks: an Ethernet network and a high-performance switch. Each node uses two separate interface cards to connect to these networks. There are two communication protocols: a standard IP and a *user-space* (US) protocol; each can run on either network. If either network or protocol fails, the other network or protocol can take over.

2.3.2.7 N-Version Programming to Enhance Software Reliability

A common isolated-redundancy approach to constructing a mission-critical software system is called *N-version programming*. The software is implemented by *N* isolated teams who may not even know the others exist. Different teams are asked to implement the software using different algorithms, programming languages, environment tools, and even platforms. In a fault-tolerant system, the

N versions all run simultaneously and their results are constantly compared. If the results differ, the system is notified that a fault has occurred. But because of isolated redundancy, it is extremely unlikely that the fault will cause a majority of the *N* versions to fail at the same time. So the system continues working, with the correct result generated by majority voting. In a highly available but less mission-critical system, only one version needs to run at a time. Each version has a built-in self-test capability. When one version fails, another version can take over.

2.3.3 Fault-Tolerant Cluster Configurations

The cluster solution was targeted to provide availability support for two server nodes with three ascending levels of availability: *hot standby*, *active takeover*, and *fault-tolerant*. In this section, we will consider the *recovery time*, *failback feature*, and *node activeness*. The level of availability increases from standby to active and fault-tolerant cluster configurations. The shorter is the recovery time, the higher is the cluster availability. *Failback* refers to the ability of a recovered node returning to normal operation after repair or maintenance. *Activeness* refers to whether the node is used in active work during normal operation.

- **Hot standby server clusters** In a *hot standby* cluster, only the primary node is actively doing all the useful work normally. The standby node is powered on (hot) and running some monitoring programs to communicate heartbeat signals to check the status of the primary node, but is not actively running other useful workloads. The primary node must mirror any data to shared disk storage, which is accessible by the standby node. The standby node requires a second copy of data.
- **Active-takeover clusters** In this case, the architecture is symmetric among multiple server nodes. Both servers are primary, doing useful work normally. Both failover and failback are often supported on both server nodes. When a node fails, the user applications fail over to the available node in the cluster. Depending on the time required to implement the failover, users may experience some delays or may lose some data that was not saved in the last checkpoint.
- **Failover cluster** This is probably the most important feature demanded in current clusters for commercial applications. When a component fails, this technique allows the remaining system to take over the services originally provided by the failed component. A failover mechanism must provide several functions, such as *failure diagnosis*, *failure notification*, and *failure recovery*. Failure diagnosis refers to the detection of a failure and the location of the failed component that caused the failure. A commonly used technique is *heartbeat*, whereby the cluster nodes send out a stream of heartbeat messages to one another. If the system does not receive the stream of heartbeat messages from a node, it can conclude that either the node or the network connection has failed.

Example 2.8 Failure Diagnosis and Recovery in a Dual-Network Cluster

A cluster uses two networks to connect its nodes. One node is designated as the *master node*. Each node has a *heartbeat daemon* that periodically (every 10 seconds) sends a heartbeat message to the master

node through both networks. The master node will detect a failure if it does not receive messages for a beat (10 seconds) from a node and will make the following diagnoses:

- A node's connection to one of the two networks failed if the master receives a heartbeat from the node through one network but not the other.
- The node failed if the master does not receive a heartbeat through either network. It is assumed that the chance of both networks failing at the same time is negligible.

The failure diagnosis in this example is simple, but it has several pitfalls. What if the master node fails? Is the 10-second heartbeat period too long or too short? What if the heartbeat messages are dropped by the network (e.g., due to network congestion)? Can this scheme accommodate hundreds of nodes? Practical HA systems must address these issues. A popular trick is to use the heartbeat messages to carry load information so that when the master receives the heartbeat from a node, it knows not only that the node is alive, but also the resource utilization status of the node. Such load information is useful for load balancing and job management.

Once a failure is diagnosed, the system notifies the components that need to know the failure event. Failure notification is needed because the master node is not the only one that needs to have this information. For instance, in case of the failure of a node, the DNS needs to be told so that it will not connect more users to that node. The resource manager needs to reassign the workload and to take over the remaining workload on that node. The system administrator needs to be alerted so that she can initiate proper actions to repair the node.

2.3.3.1 Recovery Schemes

Failure recovery refers to the actions needed to take over the workload of a failed component. There are two types of recovery techniques. In *backward recovery*, the processes running on a cluster periodically save a consistent state (called a *checkpoint*) to a stable storage. After a failure, the system is reconfigured to isolate the failed component, restores the previous checkpoint, and resumes normal operation. This is called *rollback*.

Backward recovery is relatively easy to implement in an application-independent, portable fashion, and has been widely used. However, rollback implies wasted execution. If execution time is crucial, such as in real-time systems where the rollback time cannot be tolerated, a *forward recovery* scheme should be used. With such a scheme, the system is not rolled back to the previous checkpoint upon a failure. Instead, the system utilizes the failure diagnosis information to reconstruct a valid system state and continues execution. Forward recovery is application-dependent and may need extra hardware.

Example 2.9 MTTF, MTTR, and Failure Cost Analysis

Consider a cluster that has little availability support. Upon a node failure, the following sequence of events takes place:

1. The entire system is shut down and powered off.
2. The faulty node is replaced if the failure is in hardware.
3. The system is powered on and rebooted.
4. The user application is reloaded and rerun from the start.

Assume one of the cluster nodes fails every 100 hours. Other parts of the cluster never fail. Steps 1 through 3 take two hours. On average, the mean time for step 4 is two hours. What is the availability of the cluster? What is the yearly failure cost if each one-hour downtime costs $82,500?

Solution: The cluster's MTTF is 100 hours; the MTTR is 2 + 2 = 4 hours. According to Table 2.5, the availability is 100/104 = 96.15 percent. This corresponds to 337 hours of downtime in a year, and the failure cost is $82500 × 337, that is, more than $27 million.

Example 2.10 Availability and Cost Analysis of a Cluster of Computers

Repeat Example 2.9, but assume that the cluster now has much increased availability support. Upon a node failure, its workload automatically fails over to other nodes. The failover time is only six minutes. Meanwhile, the cluster has *hot swap* capability: The faulty node is taken off the cluster, repaired, replugged, and rebooted, and it rejoins the cluster, all without impacting the rest of the cluster. What is the availability of this ideal cluster, and what is the yearly failure cost?

Solution: The cluster's MTTF is still 100 hours, but the MTTR is reduced to 0.1 hours, as the cluster is available while the failed node is being repaired. From to Table 2.5, the availability is 100/100.5 = 99.9 percent. This corresponds to 8.75 hours of downtime per year, and the failure cost is $82,500, a 27M/722K = 38 times reduction in failure cost from the design in Example 3.8.

2.3.4 Checkpointing and Recovery Techniques

Checkpointing and recovery are two techniques that must be developed hand in hand to enhance the availability of a cluster system. We will start with the basic concept of checkpointing. This is the process of periodically saving the state of an executing program to stable storage, from which the system can recover after a failure. Each program state saved is called a *checkpoint*. The disk file that contains the saved state is called the *checkpoint file*. Although all current checkpointing software saves program states in a disk, research is underway to use node memories in place of stable storage in order to improve performance.

Checkpointing techniques are useful not only for availability, but also for program debugging, process migration, and load balancing. Many job management systems and some operating systems support checkpointing to a certain degree. The Web Resource contains pointers to numerous checkpoint-related web sites, including some public domain software such as Condor and Libckpt. Here we will present the important issues for the designer and the user of checkpoint software. We will first consider the issues that are common to both sequential and parallel programs, and then we will discuss the issues pertaining to parallel programs.

2.3.4.1 Kernel, Library, and Application Levels

Checkpointing can be realized by the operating system at the *kernel level*, where the OS transparently checkpoints and restarts processes. This is ideal for users. However, checkpointing is not supported in most operating systems, especially for parallel programs. A less transparent approach links the user code with a checkpointing library in the user space. Checkpointing and restarting are

handled by this runtime support. This approach is used widely because it has the advantage that user applications do not have to be modified.

A main problem is that most current checkpointing libraries are static, meaning the application source code (or at least the object code) must be available. It does not work if the application is in the form of executable code. A third approach requires the user (or the compiler) to insert checkpointing functions in the application; thus, the application has to be modified, and the transparency is lost. However, it has the advantage that the user can specify where to checkpoint. This is helpful to reduce checkpointing overhead. Checkpointing incurs both time and storage overheads.

2.3.4.2 Checkpoint Overheads

During a program's execution, its states may be saved many times. This is denoted by the time consumed to save one checkpoint. The storage overhead is the extra memory and disk space required for checkpointing. Both time and storage overheads depend on the size of the checkpoint file. The overheads can be substantial, especially for applications that require a large memory space. A number of techniques have been suggested to reduce these overheads.

2.3.4.3 Choosing an Optimal Checkpoint Interval

The time period between two checkpoints is called the *checkpoint interval*. Making the interval larger can reduce checkpoint time overhead. However, this implies a longer computation time after a failure. Wong and Franklin [28] derived an expression for optimal checkpoint interval as illustrated in Figure 2.20.

$$\textit{Optimal checkpoint interval} = \text{Square root}\,(MTTF \times t_c)/h \tag{2.2}$$

Here, MTTF is the system's *mean time to failure*. This MTTF accounts the time consumed to save one checkpoint, and h is the average percentage of normal computation performed in a checkpoint interval before the system fails. The parameter h is always in the range. After a system is restored, it needs to spend $h \times$ (checkpoint interval) time to recompute.

2.3.4.4 Incremental Checkpoint

Instead of saving the full state at each checkpoint, an *incremental checkpoint* scheme saves only the portion of the state that is changed from the previous checkpoint. However, care must be taken regarding *old* checkpoint files. In full-state checkpointing, only one checkpoint file needs to be kept on disk. Subsequent checkpoints simply overwrite this file. With incremental checkpointing, old files needed to be kept, because a state may span many files. Thus, the total storage requirement is larger.

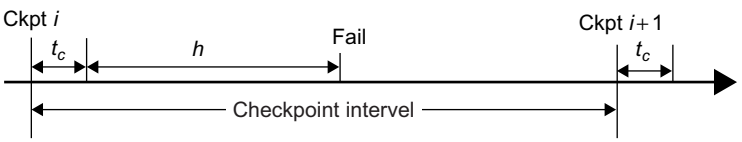


FIGURE 2.20

Time parameters between two checkpoints.

(*Courtesy of Hwang and Xu [14]*)

2.3.4.5 Forked Checkpointing

Most checkpoint schemes are blocking in that the normal computation is stopped while checkpointing is in progress. With enough memory, checkpoint overhead can be reduced by making a copy of the program state in memory and invoking another asynchronous thread to perform the checkpointing concurrently. A simple way to overlap checkpointing with computation is to use the UNIX *fork( )* system call. The forked child process duplicates the parent process's address space and checkpoints it. Meanwhile, the parent process continues execution. Overlapping is achieved since checkpointing is disk-I/O intensive. A further optimization is to use the copy-on-write mechanism.

2.3.4.6 User-Directed Checkpointing

The checkpoint overheads can sometimes be substantially reduced if the user inserts code (e.g., library or system calls) to tell the system when to save, what to save, and what not to save. What should be the exact contents of a checkpoint? It should contain just enough information to allow a system to recover. The state of a process includes its data state and control state. For a UNIX process, these states are stored in its address space, including the text (code), the data, the stack segments, and the process descriptor. Saving and restoring the full state is expensive and sometimes impossible.

For instance, the process ID and the parent process ID are not restorable, nor do they need to be saved in many applications. Most checkpointing systems save a partial state. For instance, the code segment is usually not saved, as it does not change in most applications. What kinds of applications can be checkpointed? Current checkpoint schemes require programs to be *well behaved*, the exact meaning of which differs in different schemes. At a minimum, a well-behaved program should not need the exact contents of state information that is not restorable, such as the numeric value of a process ID.

2.3.4.7 Checkpointing Parallel Programs

We now turn to checkpointing parallel programs. The state of a parallel program is usually much larger than that of a sequential program, as it consists of the set of the states of individual processes, plus the state of the communication network. Parallelism also introduces various timing and consistency problems.

Example 2.11 Checkpointing a Parallel Program

Figure 2.21 illustrates checkpointing of a three-process parallel program. The arrows labeled *x*, *y*, and *z* represent point-to-point communication among the processes. The three thick lines labeled *a*, *b*, and *c* represent three *global snapshots* (or simply *snapshots*), where a global snapshot is a set of checkpoints

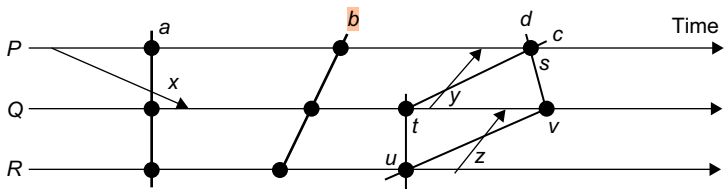


FIGURE 2.21

Consistent and inconsistent checkpoints in a parallel program.

(*Courtesy of Hwang and Xu [14]*)

(represented by dots), one from every process. In addition, some communication states may need to be saved. The intersection of a snapshot line with a process's time line indicates where the process should take a (local) checkpoint. Thus, the program's snapshot *c* consists of three local checkpoints: *s*, *t*, *u* for processes *P*, *Q*, and *R*, respectively, plus saving the communication *y*.

2.3.4.8 Consistent Snapshot

A global snapshot is called *consistent* if there is no message that is received by the checkpoint of one process, but not yet sent by another process. Graphically, this corresponds to the case that no arrow crosses a snapshot line from right to left. Thus, snapshot *a* is consistent, because arrow *x* is from left to right. But snapshot *c* is inconsistent, as *y* goes from right to left. To be consistent, there should not be any *zigzag path* between two checkpoints [20]. For instance, checkpoints *u* and *s* cannot belong to a consistent global snapshot. A stronger consistency requires that no arrows cross the snapshot. Thus, only snapshot *b* is consistent in Figure 2.23.

2.3.4.9 Coordinated versus Independent Checkpointing

Checkpointing schemes for parallel programs can be classified into two types. In *coordinated checkpointing* (also called *consistent checkpointing*), the parallel program is frozen, and all processes are checkpointed at the same time. In *independent checkpointing*, the processes are checkpointed independent of one another. These two types can be combined in various ways. Coordinated checkpointing is difficult to implement and tends to incur a large overhead. Independent checkpointing has a small overhead and can utilize existing checkpointing schemes for sequential programs.

2.4 CLUSTER JOB AND RESOURCE MANAGEMENT

This section covers various scheduling methods for executing multiple jobs on a clustered system. The LSF is described as middleware for cluster computing. MOSIX is introduced as a distributed OS for managing resources in large-scale clusters or in clouds.

2.4.1 Cluster Job Scheduling Methods

Cluster jobs may be scheduled to run at a specific time (*calendar scheduling*) or when a particular event happens (*event scheduling*). Table 2.6 summarizes various schemes to resolve job scheduling issues on a cluster. Jobs are scheduled according to priorities based on submission time, resource nodes, execution time, memory, disk, job type, and user identity. With *static priority*, jobs are assigned priorities according to a predetermined, fixed scheme. A simple scheme is to schedule jobs in a first-come, first-serve fashion. Another scheme is to assign different priorities to users. With *dynamic priority*, the priority of a job may change over time.

Three schemes are used to share cluster nodes. In the *dedicated mode*, only one job runs in the cluster at a time, and at most, one process of the job is assigned to a node at a time. The single job runs until completion before it releases the cluster to run other jobs. Note that even in the dedicated mode, some nodes may be reserved for system use and not be open to the user job. Other than that, all cluster resources are devoted to run a single job. This may lead to poor system utilization. The job resource

Table 2.6 Job Scheduling Issues and Schemes for Cluster Nodes

Issue	Scheme	Key Problems
Job priority	Nonpreemptive	Delay of high-priority jobs
	Preemptive	Overhead, implementation
Resource required	Static	Load imbalance
	Dynamic	Overhead, implementation
Resource sharing	Dedicated	Poor utilization
	Space sharing	Tiling, large job
Scheduling	Time sharing	Process-based job control with context switch overhead
	Independent	Severe slowdown
	Gang scheduling	Implementation difficulty
Competing with foreign (local) jobs	Stay	Local job slowdown
	Migrate	Migration threshold, migration overhead

requirement can be *static* or *dynamic*. Static scheme fixes the number of nodes for a single job for its entire period. Static scheme may underutilize the cluster resource. It cannot handle the situation when the needed nodes become unavailable, such as when the workstation owner shuts down the machine.

Dynamic resource allows a job to acquire or release nodes during execution. However, it is much more difficult to implement, requiring cooperation between a running job and the Java Message Service (JMS). The jobs make asynchronous requests to the JMS to add/delete resources. The JMS needs to notify the job when resources become available. The synchrony means that a job should not be delayed (blocked) by the request/notification. Cooperation between jobs and the JMS requires modification of the programming languages/libraries. A primitive mechanism for such cooperation exists in PVM and MPI.

2.4.1.1 Space Sharing

A common scheme is to assign higher priorities to short, interactive jobs in daytime and during evening hours using *tiling*. In this *space-sharing* mode, multiple jobs can run on disjointed partitions (groups) of nodes simultaneously. At most, one process is assigned to a node at a time. Although a partition of nodes is dedicated to a job, the interconnect and the I/O subsystem may be shared by all jobs. Space sharing must solve the tiling problem and the large-job problem.

Example 2.12 Job Scheduling by Tiling over Cluster Nodes

Figure 2.22 illustrates the *tiling technique*. In Part (a), the JMS schedules four jobs in a first-come first-serve fashion on four nodes. Jobs 1 and 2 are small and thus assigned to nodes 1 and 2. Jobs 3 and 4 are parallel; each needs three nodes. When job 3 comes, it cannot run immediately. It must wait until job 2 finishes to free up the needed nodes. Tiling will increase the utilization of the nodes as shown in Figure 2.22(b). The overall execution time of the four jobs is reduced after repacking the jobs over the available nodes. This problem cannot be solved in dedicated or space-sharing modes. However, it can be alleviated by timesharing.

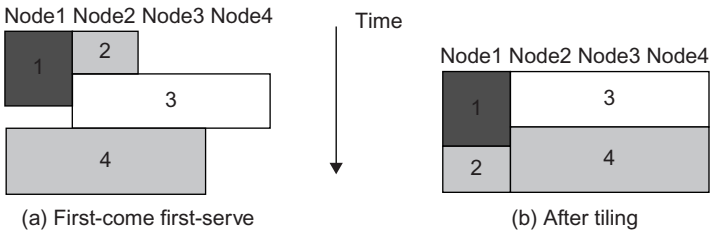


FIGURE 2.22

The tiling technique for scheduling more jobs to cluster nodes to shorten the total makespan and thus increase the job throughput.

(*Courtesy of Hwang and Xu [14]*)

2.4.1.2 Time Sharing

In the dedicated or space-sharing model, only one user process is allocated to a node. However, the system processes or daemons are still running on the same node. In the *time-sharing* mode, multiple user processes are assigned to the same node. Time sharing introduces the following parallel scheduling policies:

1. **Independent scheduling** The most straightforward implementation of time sharing is to use the operating system of each cluster node to schedule different processes as in a traditional workstation. This is called *local scheduling* or *independent scheduling*. However, the performance of parallel jobs could be significantly degraded. Processes of a parallel job need to interact. For instance, when one process wants to barrier-synchronize with another, the latter may be scheduled out. So the first process has to wait. As the second process is rescheduled, the first process may be swapped out.
2. **Gang scheduling** The *gang scheduling* scheme schedules all processes of a parallel job together. When one process is active, all processes are active. The cluster nodes are not perfectly clock-synchronized. In fact, most clusters are asynchronous systems, and are not driven by the same clock. Although we say, "All processes are scheduled to run at the same time," they do not start exactly at the same time. *Gang-scheduling skew* is the maximum difference between the time the first process starts and the time the last process starts. The execution time of a parallel job increases as the gang-scheduling skew becomes larger, leading to longer execution time. We should use a homogeneous cluster, where gang scheduling is more effective. However, gang scheduling is not yet realized in most clusters, because of implementation difficulties.
3. **Competition with foreign (local) jobs** Scheduling becomes more complicated when both cluster jobs and local jobs are running. Local jobs should have priority over cluster jobs. With one keystroke, the owner wants command of all workstation resources. There are basically two ways to deal with this situation: The cluster job can either stay in the workstation node or migrate to another idle node. A *stay scheme* has the advantage of avoiding migration cost. The cluster process can be run at the lowest priority. The workstation's cycles can be divided into three portions, for kernel processes, local processes, and cluster processes. However, to stay slows down both the local and the cluster jobs, especially when the cluster job is a load-balanced parallel job that needs frequent synchronization and communication. This leads to the migration approach to flow the jobs around available nodes, mainly for balancing the workload.

2.4.2 Cluster Job Management Systems

Job management is also known as *workload management*, *load sharing*, or *load management*. We will first discuss the basic issues facing a job management system and summarize the available software packages. A *Job Management System (JMS)* should have three parts:

- A *user server* lets the user submit jobs to one or more queues, specify resource requirements for each job, delete a job from a queue, and inquire about the status of a job or a queue.
- A *job scheduler* performs job scheduling and queuing according to job types, resource requirements, resource availability, and scheduling policies.
- A *resource manager* allocates and monitors resources, enforces scheduling policies, and collects accounting information.

2.4.2.1 JMS Administration

The functionality of a JMS is often distributed. For instance, a user server may reside in each host node, and the resource manager may span all cluster nodes. However, the administration of a JMS should be *centralized*. All configuration and log files should be maintained in one location. There should be a single user interface to use the JMS. It is undesirable to force the user to run PVM jobs through one software package, MPI jobs through another, and HPF jobs through yet another.

The JMS should be able to dynamically reconfigure the cluster with minimal impact on the running jobs. The administrator's prologue and epilogue scripts should be able to run before and after each job for security checking, accounting, and cleanup. Users should be able to cleanly kill their own jobs. The administrator or the JMS should be able to cleanly suspend or kill any job. *Clean* means that when a job is suspended or killed, all its processes must be included. Otherwise, some "orphan" processes are left in the system, which wastes cluster resources and may eventually render the system unusable.

2.4.2.2 Cluster Job Types

Several types of jobs execute on a cluster. *Serial jobs* run on a single node. *Parallel jobs* use multiple nodes. *Interactive jobs* are those that require fast turnaround time, and their input/output is directed to a terminal. These jobs do not need large resources, and users expect them to execute immediately, not to wait in a queue. *Batch jobs* normally need more resources, such as large memory space and long CPU time. But they do not need immediate responses. They are submitted to a job queue to be scheduled to run when the resource becomes available (e.g., during off hours).

While both interactive and batch jobs are managed by the JMS, *foreign jobs* are created outside the JMS. For instance, when a network of workstations is used as a cluster, users can submit interactive or batch jobs to the JMS. Meanwhile, the owner of a workstation can start a foreign job at any time, which is not submitted through the JMS. Such a job is also called a *local job*, as opposed to *cluster jobs* (interactive or batch, parallel or serial) that are submitted through the JMS of the cluster. The characteristic of a local job is fast response time. The owner wants all resources to execute his job, as though the cluster jobs did not exist.

2.4.2.3 Characteristics of a Cluster Workload

To realistically address job management issues, we must understand the workload behavior of clusters. It may seem ideal to characterize workload based on long-time operation data on real clusters. The parallel workload traces include both development and production jobs. These traces are then

fed to a simulator to generate various statistical and performance results, based on different sequential and parallel workload combinations, resource allocations, and scheduling policies. The following workload characteristics are based on a NAS benchmark experiment. Of course, different workloads may have variable statistics.

- Roughly half of parallel jobs are submitted during regular working hours. Almost 80 percent of parallel jobs run for three minutes or less. Parallel jobs running longer than 90 minutes account for 50 percent of the total time.
- The sequential workload shows that 60 percent to 70 percent of workstations are available to execute parallel jobs at any time, even during peak daytime hours.
- On a workstation, 53 percent of all idle periods are three minutes or less, but 95 percent of idle time is spent in periods of time that are 10 minutes or longer.
- A 2:1 rule applies, which says that a network of 64 workstations, with proper JMS software, can sustain a 32-node parallel workload in addition to the original sequential workload. In other words, clustering gives a supercomputer half of the cluster size for free!

2.4.2.4 Migration Schemes

A *migration scheme* must consider the following three issues:

- **Node availability** This refers to node availability for job migration. The Berkeley NOW project has reported such opportunity does exist in university campus environment. Even during peak hours, 60 percent of workstations in a cluster are available at Berkeley campus.
- **Migration overhead** What is the effect of the migration overhead? The migration time can significantly slow down a parallel job. It is important to reduce the migration overhead (e.g., by improving the communication subsystem) or to migrate only rarely. The slowdown is significantly reduced if a parallel job is run on a cluster of twice the size. For instance, for a 32-node parallel job run on a 60-node cluster, the slowdown caused by migration is no more than 20 percent, even when the migration time is as long as three minutes. This is because more nodes are available, and thus the migration demand is less frequent.
- **Recruitment threshold** What should be the recruitment threshold? In the worst scenario, right after a process migrates to a node, the node is immediately claimed by its owner. Thus, the process has to migrate again, and the cycle continues. The recruitment threshold is the amount of time a workstation stays unused before the cluster considers it an idle node.

2.4.2.5 Desired Features in A JMS

Here are some features that have been built in some commercial JMSes in cluster computing applications:

- Most support heterogeneous Linux clusters. All support parallel and batch jobs. However, Connect:Queue does not support interactive jobs.
- Enterprise cluster jobs are managed by the JMS. They will impact the owner of a workstation in running local jobs. However, NQE and PBS allow the impact to be adjusted. In DQS, the impact can be configured to be minimal.
- All packages offer some kind of load-balancing mechanism to efficiently utilize cluster resources. Some packages support checkpointing.
- Most packages cannot support dynamic process migration. They support static migration: A process can be dispatched to execute on a remote node when the process is first created.

However, once it starts execution, it stays in that node. A package that does support dynamic process migration is Condor.

- All packages allow dynamic suspension and resumption of a user job by the user or by the administrator. All packages allow resources (e.g., nodes) to be dynamically added to or deleted.
- Most packages provide both a command-line interface and a graphical user interface. Besides UNIX security mechanisms, most packages use the Kerberos authentication system.

2.4.3 Load Sharing Facility (LSF) for Cluster Computing

LSF is a commercial workload management system from Platform Computing [29]. LSF emphasizes job management and load sharing on both parallel and sequential jobs. In addition, it supports checkpointing, availability, load migration, and SSI. LSF is highly scalable and can support a cluster of thousands of nodes. LSF has been implemented for various UNIX and Windows/NT platforms. Currently, LSF is being used not only in clusters but also in grids and clouds.

2.4.3.1 LSF Architecture

LSF supports most UNIX platforms and uses the standard IP for JMS communication. Because of this, it can convert a heterogeneous network of UNIX computers into a cluster. There is no need to change the underlying OS kernel. The end user utilizes the LSF functionalities through a set of utility commands. PVM and MPI are supported. Both a command-line interface and a GUI are provided. LSF also offers skilled users an API that is a runtime library called *LSLIB (load sharing library)*. Using LSLIB explicitly requires the user to modify the application code, whereas using the utility commands does not. Two LSF daemons are used on each server in the cluster. The *load information managers (LIMs)* periodically exchange load information. The *remote execution server (RES)* executes remote tasks.

2.4.3.2 LSF Utility Commands

A cluster node may be a single-processor host or an SMP node with multiple processors, but always runs with only a single copy of the operating system on the node. Here are interesting features built into the LSF facilities:

- LSF supports all four combinations of interactive, batch, sequential, and parallel jobs. A job that is not executed through LSF is called a *foreign job*. A *server node* is one which can execute LSF jobs. A *client node* is one that can initiate or submit LSF jobs but cannot execute them. Only the resources on the server nodes can be shared. Server nodes can also initiate or submit LSF jobs.
- LSF offers a set of tools (*lstools*) to get information from LSF and to run jobs remotely. For instance, *lshosts* lists the static resources (discussed shortly) of every server node in the cluster. The command *lsrun* executes a program on a remote node.
- When a user types the command line *%lsrun-R 'swp>100' myjob* at a client node, the application *myjob* will be automatically executed on the most lightly loaded server node that has an available swap space greater than 100 MB.
- The *lsbatch* utility allows users to submit, monitor, and execute batch jobs through LSF. This utility is a load-sharing version of the popular UNIX command interpreter *tcsh*. Once a user enters the *lstcsh* shell, every command issued will be automatically executed on a suitable node. This is done transparently: The user sees a shell exactly like a *tcsh* running on the local node.

- The *lsmake* utility is a parallel version of the UNIX make utility, allowing a makefile to be processed in multiple nodes simultaneously.

Example 2.13 Application of the LSF on a Cluster of Computers

Suppose a cluster consists of eight expensive server nodes and 100 inexpensive client nodes (workstations or PCs). The server nodes are expensive due to better hardware and software, including application software. A license is available to install a FORTRAN compiler and a CAD simulation package, both valid for up to four users. Using a JMS such as LSF, all the hardware and software resources of the server nodes are made available to the clients transparently.

A user sitting in front of a client's terminal feels as though the client node has all the software and speed of the servers locally. By typing *lsmake my.makefile*, the user can compile his source code on up to four servers. LSF selects the nodes with the least amount of load. Using LSF also benefits resource utilization. For instance, a user wanting to run a CAD simulation can submit a batch job. LSF will schedule the job as soon as the software becomes available.

2.4.4 MOSIX: An OS for Linux Clusters and Clouds

MOSIX is a distributed operating system that was developed at Hebrew University in 1977. Initially, the system extended BSD/OS system calls for resource sharing in Pentium clusters. In 1999, the system was redesigned to run on Linux clusters built with x86 platforms. The MOSIX project is still active as of 2011, with 10 versions released over the years. The latest version, MOSIX2, is compatible with Linux 2.6.

2.4.4.1 MOXIS2 for Linux Clusters

MOSIX2 runs as a virtualization layer in the Linux environment. This layer provides SSI to users and applications along with runtime Linux support. The system runs applications in remote nodes as though they were run locally. It supports both sequential and parallel applications, and can discover resources and migrate software processes transparently and automatically among Linux nodes. MOSIX2 can also manage a Linux cluster or a grid of multiple clusters.

Flexible management of a grid allows owners of clusters to share their computational resources among multiple cluster owners. Each cluster can still preserve its autonomy over its own clusters and its ability to disconnect its nodes from the grid at any time. This can be done without disrupting the running programs. A MOSIX-enabled grid can extend indefinitely as long as trust exists among the cluster owners. The condition is to guarantee that guest applications cannot be modified while running in remote clusters. Hostile computers are not allowed to connect to the local network.

2.4.4.2 SSI Features in MOSIX2

The system can run in *native mode* or as a VM. In native mode, the performance is better, but it requires that you modify the base Linux kernel, whereas a VM can run on top of any unmodified OS that supports virtualization, including Microsoft Windows, Linux, and Mac OS X. The system

is most suitable for running compute-intensive applications with low to moderate amounts of I/O. Tests of MOSIX2 show that the performance of several such applications over a 1 GB/second campus grid is nearly identical to that of a single cluster. Here are some interesting features of MOSIX2:

- Users can log in on any node and do not need to know where their programs run.
- There is no need to modify or link applications with special libraries.
- There is no need to copy files to remote nodes, thanks to automatic resource discovery and workload distribution by process migration.
- Users can load-balance and migrate processes from slower to faster nodes and from nodes that run out of free memory.
- Sockets are migratable for direct communication among migrated processes.
- The system features a secure runtime environment (sandbox) for guest processes.
- The system can run batch jobs with checkpoint recovery along with tools for automatic installation and configuration scripts.

2.4.4.3 Applications of MOSIX for HPC

The MOSIX is a research OS for HPC cluster, grid, and cloud computing. The system's designers claim that MOSIX offers efficient utilization of wide-area grid resources through automatic resource discovery and load balancing. The system can run applications with unpredictable resource requirements or runtimes by running long processes, which are automatically sent to grid nodes. The system can also combine nodes of different capacities by migrating processes among nodes based on their load index and available memory.

MOSIX became proprietary software in 2001. Application examples include scientific computations for genomic sequence analysis, molecular dynamics, quantum dynamics, nanotechnology and other parallel HPC applications; engineering applications including CFD, weather forecasting, crash simulations, oil industry simulations, ASIC design, and pharmaceutical design; and cloud applications such as for financial modeling, rendering farms, and compilation farms.

Example 2.14 Memory-Ushering Algorithm Using MOSIX versus PVM

Memory ushering is practiced to borrow the main memory of a remote cluster node, when the main memory on a local node is exhausted. The remote memory access is done by process migration instead of paging or swapping to local disks. The ushering process can be implemented with PVM commands or it can use MOSIX process migration. In each execution, an average memory chunk can be assigned to the nodes using PVM. Figure 2.23 shows the execution time of the memory algorithm using PVM compared with the use of the MOSIX routine.

For a small cluster of eight nodes, the execution times are closer. When the cluster scales to 32 nodes, the MOSIX routine shows a 60 percent reduction in ushering time. Furthermore, MOSIX performs almost the same when the cluster size increases. The PVM ushering time increases monotonically by an average of 3.8 percent per node increase, while MOSIX consistently decreases by 0.4 percent per node increase. The reduction in time results from the fact that the memory and load-balancing algorithms of MOSIX are more scalable than PVM.

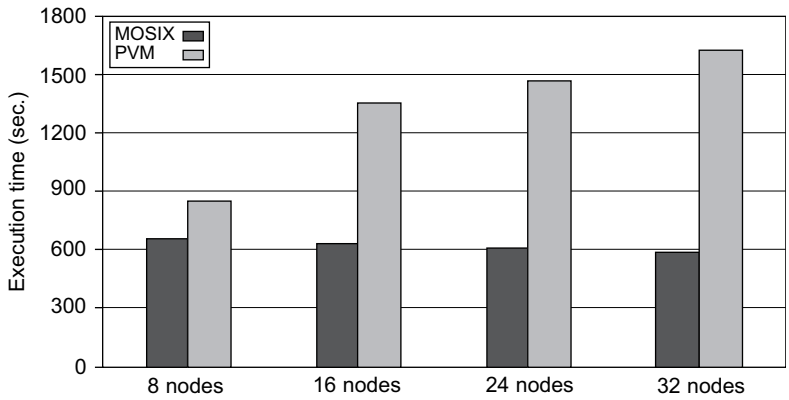


FIGURE 2.23

Performance of the memory-ushering algorithm using MOSIX versus PVM.

(*Courtesy of A. Barak and O. La'adan [5]*)

2.5 CASE STUDIES OF TOP SUPERCOMPUTER SYSTEMS

This section reviews three top supercomputers that have been singled out as winners in the Top 500 List for the years 2008–2010. The IBM Roadrunner was the world's first petaflops computer, ranked No. 1 in 2008. Subsequently, the Cray XT5 Jaguar became the top system in 2009. In November 2010, China's Tianhe-1A became the fastest system in the world. All three systems are Linux cluster-structured with massive parallelism in term of large number of compute nodes that can execute concurrently.

2.5.1 Tianhe-1A: The World Fastest Supercomputer in 2010

In November 2010, the Tianhe-1A was unveiled as a hybrid supercomputer at the 2010 ACM Supercomputing Conference. This system demonstrated a sustained speed of 2.507 Pflops in Linpack Benchmark testing runs and thus became the No. 1 supercomputer in the 2010 Top 500 list. The system was built by the National University of Defense Technology (NUDT) and was installed in August 2010 at the National Supercomputer Center (NSC), Tianjin, in northern China (www.nscc.tj.gov.cn). The system is intended as an open platform for research and education. Figure 2.24 shows the Tianh-1A system installed at NSC.

2.5.1.1 Architecture of Tianhe-1A

Figure 2.25 shows the abstract architecture of the Tianhe-1A system. The system consists of five major components. The compute subsystem houses all the CPUs and GPUs on 7,168 compute nodes. The service subsystem comprises eight operation nodes. The storage subsystem has a large number of shared disks. The monitoring and diagnosis subsystem is used for control and I/O operations. The communication subsystem is composed of switches for connecting to all functional subsystems.

2.5.1.2 Hardware Implementation

This system is equipped with 14,336 six-core Xeon E5540/E5450 processors running 2.93 GHz with 7,168 NVIDIA Tesla M2050s. It has 7,168 *compute nodes*, each composed of two Intel Xeon X5670 (Westmere) processors at 2.93 GHz, six cores per socket, and one NVIDIA M2050 GPU

FIGURE 2.24

The Tianhe-1A system built by the National University of Defense Technology and installed at the National Supercomputer Center, Tianjin, China, in 2010 [11].

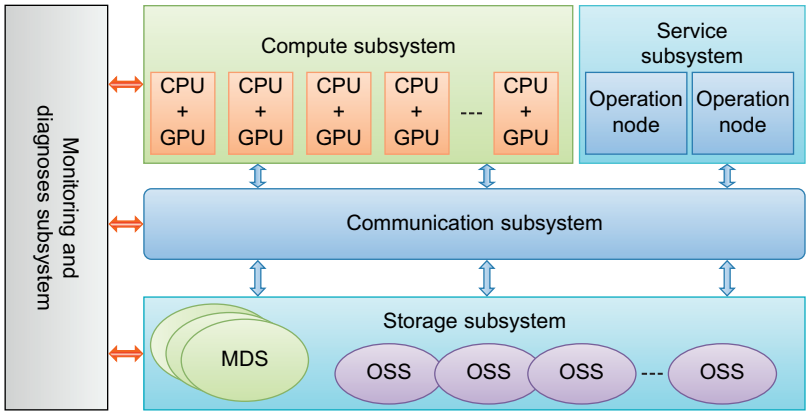


FIGURE 2.25

Abstract architecture of the Tianhe-1A system.

connected via PCI-E. A blade has two nodes and is 2U in height (Figure 2.25). The complete system has 14,336 Intel sockets (Westmere) plus 7,168 NVIDIA Fermi boards plus 2,048 Galaxy sockets (the Galaxy processor-based nodes are used as frontend processing for the system). A compute node has two Intel sockets plus a Fermi board plus 32 GB of memory.

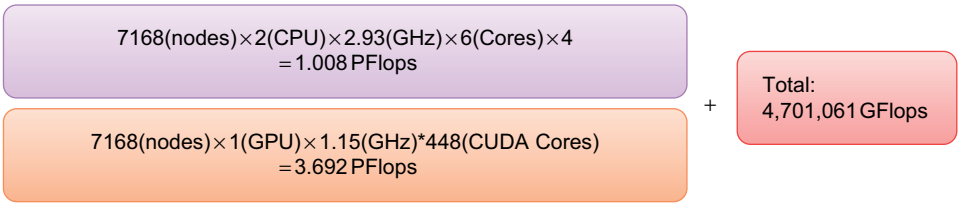


FIGURE 2.26

Calculation of the theoretical peak speed of Tianhe-1A system.

The total system has a theoretical peak of 4.7 Pflops/second as calculated in Figure 2.26. Note that there are 448 CUDA cores in each GPU node. The peak speed is achieved through 14,236 Xeon CPUs (with 380,064 cores) and 7,168 Tesla GPUs (with 448 CUDA cores per node and 3,496,884 CUDA cores in total). There are 3,876,948 processing cores in both the CPU and GPU chips. An operational node has two eight-core Galaxy chips (1 GHz, SPARC architecture) plus 32 GB of memory. The Tianhe-1A system is packaged in 112 compute cabinets, 12 storage cabinets, six communications cabinets, and eight I/O cabinets.

The operation nodes are composed of two eight-core Galaxy FT-1000 chips. These processors were designed by NUDT and run at 1 GHz. The theoretical peak for the eight-core chip is 8 Gflops/second. The complete system has 1,024 of these operational nodes with each having 32 GB of memory. These operational nodes are intended to function as service nodes for job creation and submission. They are not intended as general-purpose computational nodes. Their speed is excluded from the calculation of the peak or sustained speed. The peak speed of the Tianhe-1A is calculated as 3.692 Pflops [11]. It uses 7,168 compute nodes (with 448 CUDA cores/GPU/compute node) in parallel with 14,236 CPUs with six cores in four subsystems.

The system has total disk storage of 2 petabytes implemented with a Lustre clustered file system. There are 262 terabytes of main memory distributed in the cluster system. The Tianhe-1A epitomizes modern heterogeneous CPU/GPU computing, enabling significant achievements in performance, size, and power. The system would require more than 50,000 CPUs and twice as much floor space to deliver the same performance using CPUs alone. A 2.507-petaflop system built entirely with CPUs would consume at least 12 megawatts, which is three times more power than what the Tianhe-1A consumes.

2.5.1.3 ARCH Fat-Tree Interconnect

The high performance of the Tianhe-1A is attributed to a customed-designed ARCH interconnect by the NUDT builder. This ARCH is built with the InfiniBand DDR 4X and 98 TB of memory. It assumes a fat-tree architecture as shown in Figure 2.27. The bidirectional bandwidth is 160 Gbps, about twice the bandwidth of the QDR InfiniBand network over the same number of nodes. The ARCH has a latency for a node hop of 1.57 microseconds, and an aggregated bandwidth of 61 Tb/second. At the first stage of the ARCH fat tree, 16 nodes are connected by a 16-port switching board. At the second stage, all parts are connects to eleven 384-port switches. The router and network interface chips are designed by the NUDT team.

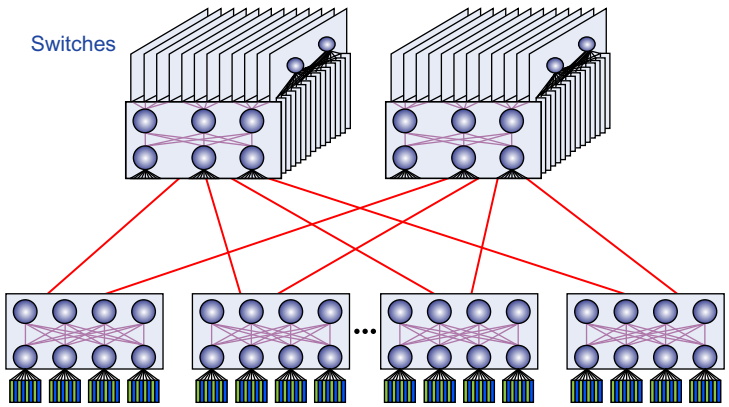


FIGURE 2.27

The ARCH fat-tree interconnect in two stages of high-bandwidth switches [11].

2.5.1.4 Software Stack

The software stack on the Tianhe-1A is typical of any high-performance system. It uses Kylin Linux, an operating system developed by NUDT and successfully approved by China's 863 Hi-tech Research and Development Program office in 2006. Kylin is based on Mach and FreeBSD, is compatible with other mainstream operating systems, and supports multiple microprocessors and computers of different structures. Kylin packages include standard open source and public packages, which have been brought onto one system for easy installation. Figure 2.28 depicts the Tianhe-1A software architecture.

The system features FORTRAN, C, C++, and Java compilers from Intel (icc 11.1), CUDA, OpenMP, and MPI based on MPICH2 with custom GLEX (Galaxy Express) Channel support. The NUDT builder developed a mathematics library, which is based on Intel's MKL 10.3.1.048 and BLAS for the GPU based on NVIDIA and optimized by NUDT. In addition, a High Productive Parallel Running Environment (HPPRE) was installed. This provides a parallel toolkit based on Eclipse, which is intended to integrate all the tools for editing, debugging, and performance analysis. In addition, the designers provide workflow support for Quality of Service (QoS) negotiations and resource reservations.

2.5.1.5 Power Consumption, Space, and Cost

The power consumption of the Tianhe-1A under load is 4.04 MWatt. The system has a footprint of 700 square meters and is cooled by a closed-coupled chilled water-cooling system with forced air. The hybrid architecture consumes less power—about one-third of the 12 MW that is needed to run the system entirely with the multicore CPUs. The budget for the system is 600 million RMB (approximately $90 million); 200 million RMB comes from the Ministry of Science and Technology (MOST) and 400 million RMB is from the Tianjin local government. It takes about $20 million annually to run, maintain, and keep the system cool in normal operations.

2.5.1.6 Linpack Benchmark Results and Planned Applications

The performance of the Linpack Benchmark on October 30, 2010 was 2.566 Pflops/second on a matrix of 3,600,000 and a $N_{1/2}$ = 1,000,000. The total time for the run was 3 hours and 22 minutes.

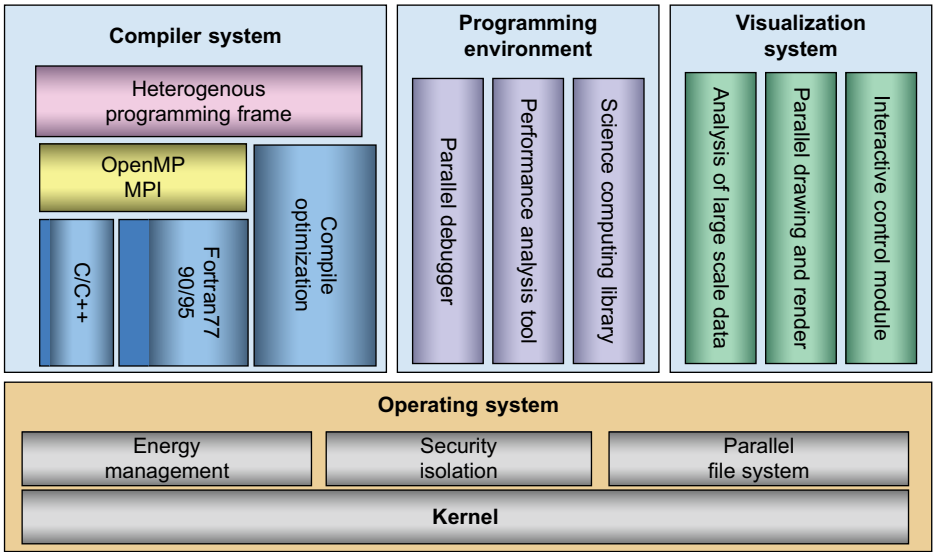


FIGURE 2.28

Software architecture of the Tianhe-1A supercomputer [11].

The system has an efficiency of 54.58 percent, which is much lower than the 75 percent efficiency achieved by Jaguar and Roadrunner. Listed below are some applications of Tianhe-1A. Most of them are specially tailored to satisfy China's national needs.

- Parallel AMR (Adaptive Mesh Refinement) method
- Parallel eigenvalue problems
- Parallel fast multipole methods
- Parallel computing models
- Gridmol computational chemistry
- ScGrid middleware, grid portal
- PSEPS parallel symmetric eigenvalue package solvers
- FMM-radar fast multipole methods on radar cross sections
- Transplant many open source software programs
- Sandstorm prediction, climate modeling, EM scattering, or cosmology
- CAD/CAE for automotive industry

2.5.2 Cray XT5 Jaguar: The Top Supercomputer in 2009

The Cray XT5 Jaguar was ranked the world's fastest supercomputer in the Top 500 list released at the ACM Supercomputing Conference in June 2010. This system became the second fastest supercomputer in the Top 500 list released in November 2010, when China's Tianhe-1A replaced the Jaguar as the No. 1 machine. This is a scalable MPP system built by Cray, Inc. The Jaguar belongs to Cray's system model XT5-HE. The system is installed at the Oak Ridge National Laboratory,

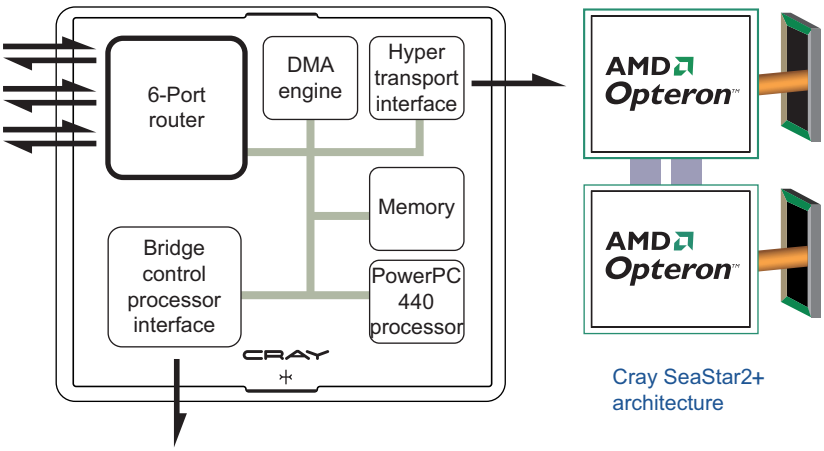


FIGURE 2.29

The interconnect SeaStar router chip design in the Cray XT5 Jaguar supercomputer.

(*Courtesy of Cray, Inc. [9] and Oak Ridge National Laboratory, United States, 2009*)

Department of Energy, in the United States. The entire Jaguar system is built with 86 cabinets. The following are some interesting architectural and operational features of the Jaguar system:

- Built with AMD six-core Opteron processors running Linux at a 2.6 GHz clock rate
- Has a total of 224,162 cores on more than 37,360 processors in 88 cabinets in four rows (there are 1,536 or 2,304 processor cores per cabinet)
- Features 8,256 compute nodes and 96 service nodes interconnected by a 3D torus network, built with Cray SeaStar2+ chips
- Attained a sustained speed, R_{max}, from the Linpack Benchmark test of 1.759 Pflops
- Largest Linpack matrix size tested recorded as N_{max} = 5,474,272 unknowns

The basic building blocks are the compute blades. The interconnect router in the SeaStar+ chip (Figure 2.29) provides six high-speed links to six neighbors in the 3D torus, as seen in Figure 2.30. The system is scalable by design from small to large configurations. The entire system has 129 TB of compute memory. In theory, the system was designed with a peak speed of $R_{peak} = 2.331$ Pflops. In other words, only 75 percent (=1.759/2.331) efficiency was achieved in Linpack experiments. The external I/O interface uses 10 Gbps Ethernet and InfiniBand links. MPI 2.1 was applied in message-passing programming. The system consumes 32–43 KW per cabinet. With 160 cabinets, the entire system consumes up to 6.950 MW. The system is cooled with forced cool air, which consumes a lot of electricity.

2.5.2.1 3D Torus Interconnect

Figure 2.30 shows the system's interconnect architecture. The Cray XT5 system incorporates a high-bandwidth, low-latency interconnect using the Cray SeaStar2+ router chips. The system is configured with XT5 compute blades with eight sockets supporting dual or quad-core Opterons. The XT5 applies a 3D torus network topology. This SeaStar2+ chip provides six high-speed network

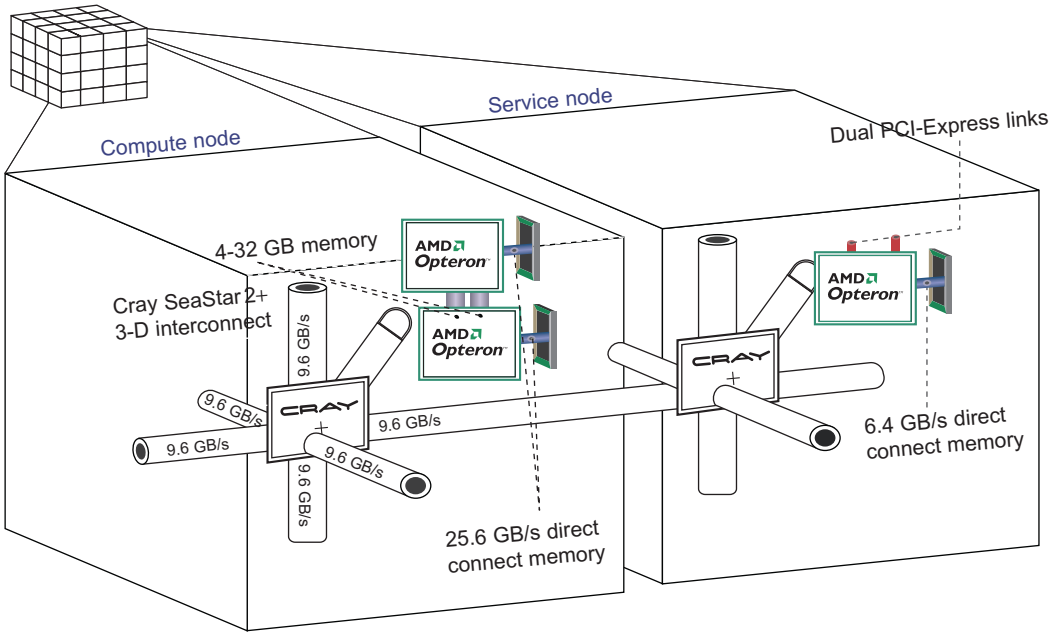


FIGURE 2.30

The 3D torus interconnect in the Cray XT5 Jaguar supercomputer.

(*Courtesy of Cray, Inc. [9] and Oak Ridge National Laboratory, United States, 2009*)

links which connect to six neighbors in the 3D torus. The peak bidirectional bandwidth of each link is 9.6 GB/second with sustained bandwidth in excess of 6 GB/second. Each port is configured with an independent router table, ensuring contention-free access for packets.

The router is designed with a reliable link-level protocol with error correction and retransmission, ensuring that message-passing traffic reliably reaches its destination without the costly timeout and retry mechanism used in typical clusters. The torus interconnect directly connects all the nodes in the Cray XT5 system, eliminating the cost and complexity of external switches and allowing for easy expandability. This allows systems to economically scale to tens of thousands of nodes—well beyond the capacity of fat-tree switches. The interconnect carries all message-passing and I/O traffic to the global file system.

2.5.2.2 Hardware Packaging

The Cray XT5 family employs an energy-efficient packaging technology, which reduces power use and thus lowers maintenance costs. The system's compute blades are packaged with only the necessary components for building an MPP with processors, memory, and interconnect. In a Cray XT5 cabinet, vertical cooling takes cold air straight from its source—the floor—and efficiently cools the processors on the blades, which are uniquely positioned for optimal airflow. Each processor also has a custom-designed heat sink depending on its position within the cabinet. Each Cray XT5 system cabinet is cooled with a single, high-efficiency ducted turbine fan. It takes 400/480VAC directly from the power grid without transformer and PDU loss.

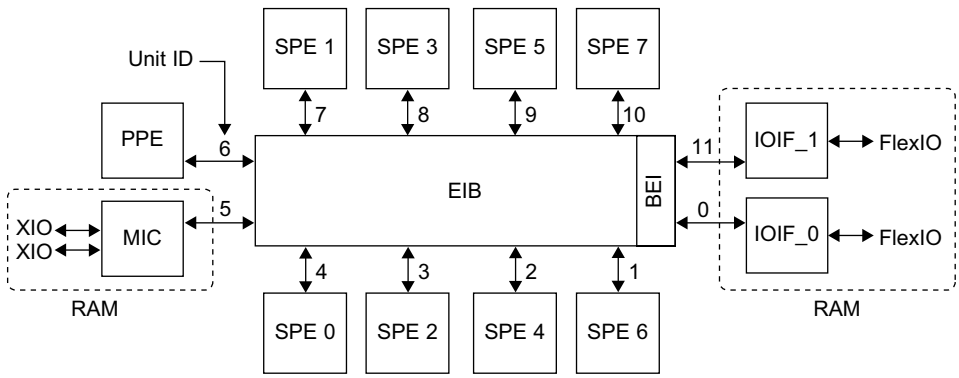
The Cray XT5 3D torus architecture is designed for superior MPI performance in HPC applications. This is accomplished by incorporating dedicated compute nodes and service nodes. Compute nodes are designed to run MPI tasks efficiently and reliably to completion. Each compute node is composed of one or two AMD Opteron microprocessors (dual or quad core) and direct attached memory, coupled with a dedicated communications resource. Service nodes are designed to provide system and I/O connectivity and also serve as login nodes from which jobs are compiled and launched. The I/O bandwidth of each compute node is designed for 25.6 GB/second performance.

2.5.3 IBM Roadrunner: The Top Supercomputer in 2008

In 2008, the IBM Roadrunner was the first general-purpose computer system in the world to reach petaflops performance. The system has a Linpack performance of 1.456 Pflops and is installed at the Los Alamos National Laboratory (LANL) in New Mexico. Subsequently, Cray's Jaguar topped the Roadrunner in late 2009. The system was used mainly to assess the decay of the U.S. nuclear arsenal. The system has a hybrid design with 12,960 IBM 3.2 GHz PowerXcell 8i CPUs (Figure 2.26) and 6,480 AMD 1.8 GHz Opteron 2210 dual-core processors. In total, the system has 122,400 cores. Roadrunner is an Opteron cluster accelerated by IBM Cell processors with eight floating-point cores.

2.5.3.1 Processor Chip and Compute Blade Design

The Cell/B.E. processors provide extraordinary compute power that can be harnessed from a single multicore chip. As shown in Figure 2.31, the Cell/B.E. architecture supports a very broad range of applications. The first implementation is a single-chip multiprocessor with nine processor elements



BEI	Cell broadband engine interface	MIC	Memory interface controller
EIB	Element interconnect bus	PPE	PowerPC processor element
FlexIO	Rambus flexIO bus	RAM	Resource allocation management
IOIF	I/O interface	SPE	Synergistic processor element
		XIO	Rambus XDR I/O (XIO) cell

FIGURE 2.31

Schematic of the IBMCell processor architecture.

(*Courtesy of IBM, http://www.redbooks.ibm.com/redpapers/pdfs/redp4477.pdf [28]*)

operating on a shared memory model. The rack is built with TriBlade servers, which are connected by an InfiniBand network. In order to sustain this compute power, the connectivity within each node consists of four PCI Express x8 links, each capable of 2 GB/s transfer rates, with a 2 μs latency. The expansion slot also contains the InfiniBand interconnect, which allows communications to the rest of the cluster. The capability of the InfiniBand interconnect is rated at 2 GB/s with a 2 μs latency.

2.5.3.2 InfiniBand Interconnect

The Roadrunner cluster was constructed hierarchically. The InfiniBand switches cluster together 18 connected units in 270 racks. In total, the cluster connects 12,960 IBM Power XCell 8i processors and 6,480 Opteron 2210 processors together with a total of 103.6 TB of RAM. This cluster complex delivers approximately 1.3 Pflops. In addition, the system's 18 Com/Service nodes deliver 4.5 Tflops using 18 InfiniBand switches. The second storage units are connected with eight InfiniBand switches. In total, 296 racks are installed in the system. The tiered architecture is constructed in two levels. The system consumes 2.35 MW power, and was the fourth most energy-efficient supercomputer built in 2009.

2.5.3.3 Message-Passing Performance

The Roadrunner uses MPI APIs to communicate with the other Opteron processors the application is running on in a typical single-program, multiple-data (SPMD) fashion. The number of compute nodes used to run the application is determined at program launch. The MPI implementation of Roadrunner is based on the open source Open MPI Project, and therefore is standard MPI. In this regard, Roadrunner applications are similar to other typical MPI applications such as those that run on the IBM Blue Gene solution. Where Roadrunner differs in the sphere of application architecture is how its Cell/B.E. accelerators are employed. At any point in the application flow, the MPI application running on each Opteron can offload computationally complex logic to its subordinate Cell/B.E. processor.

2.6 BIBLIOGRAPHIC NOTES AND HOMEWORK PROBLEMS

Cluster computing has been a hot research area since 1990. Cluster computing was pioneered by DEC and IBM as reported in Pfister [26]. His book provides a good introduction of several key concepts, including SSI and HA. Historically, milestone computer clusters include the VAXcluster running the VMS/OS in 1984, the Tandem Himalaya HA cluster (1994), and the IBM SP2 cluster in 1996. These earlier clusters were covered in [3,7,14,23,26]. In recent years, more than 85 percent of the Top 500 systems are built with cluster configurations [9,11,20,25,28,29].

Annually, IEEE and ACM hold several international conferences related to this topic. They include Cluster Computing (Cluster); Supercomputing Conference (SC); International Symposium on Parallel and Distributed Systems (IPDPS); International Conferences on Distributed Computing Systems (ICDCS); High-Performance Distributed Computing (HPDC); and Clusters, Clouds, and The Grids (CCGrid). There are also several journals related to this topic, including the Journal of Cluster Computing, Journal of Parallel and Distributed Computing (JPDC), and IEEE Transactions on Parallel and Distributed Systems (TPDS).

Cluster applications are assessed in Bader and Pennington [2]. Some figures and examples in this chapter are modified from the earlier book by Hwang and Xu [14]. Buyya has treated cluster

computing in two edited volumes [7]. Two books on Linux clusters are [20,23]. HA clusters are treated in [24]. Recent assessment of HPC interconnects can be found in [6,8,12,22]. The Google cluster interconnect was reported by Barroso, et al. [6]. GPUs for supercomputing was discussed in [10]. GPU clusters were studied in [19]. CUDA parallel programming for GPUs is treated in [31]. MOSIX/OS for cluster or grid computing is treated in [4,5,30].

Hwang, Jin, and Ho developed a distributed RAID system for achieving a single I/O space in a cluster of PCs or workstations [13–17]. More details of LSF can be found in Zhou [35]. The Top 500 list was cited from the release in June and November 2010 [25]. The material on the Tianhe-1A can be found in Dongarra [11] and on Wikipedia [29]. The IBM Blue Gene/L architecture was reported by Adiga, et al. [1] and subsequently upgraded to a newer model called the Blue Gene/P solution. The IBM Roadrunner was reported by Kevin, et al. [18] and also in Wikipedia [28]. The Cray XT5 and Jaguar systems are described in [9]. China's Nebulae supercomputer was reported in [27]. Specific cluster applications and checkpointing techniques can be found in [12,16,17,24,32,34]. Cluster applications can be found in [7,15,18,21,26,27,33,34].

Acknowledgments

This chapter is authored by Kai Hwang of USC and by Jack Dongarra of UTK jointly. Some cluster material are borrowed from the earlier book [14] by Kai Hwang of USC and Zhiwei Xu of the Chinese Academy of Sciences. Valuable suggestions to update the material were made by Rajkumar Buyya of the University of Melbourne.

References

[1] N. Adiga, et al., An overview of the blue gene/L supercomputer, in: ACM Supercomputing Conference 2002, November 2002, http://SC-2002.org/paperpdfs/pap.pap207.pdf.
[2] D. Bader, R. Pennington, Cluster computing applications, Int. J. High Perform. Comput. (May) (2001).
[3] M. Baker, et al., Cluster computing white paper. http://arxiv.org/abs/cs/0004014, January 2001.
[4] A. Barak, A. Shiloh, The MOSIX Management Systems for Linux Clusters, Multi-Clusters and Clouds. White paper, www.MOSIX.org//txt_pub.html, 2010.
[5] A. Barak, R. La'adan, The MOSIX multicomputer operating systems for high-performance cluster computing, Future Gener. Comput. Syst. 13 (1998) 361–372.
[6] L. Barroso, J. Dean, U. Holzle, Web search for a planet: The Google cluster architecture, IEEE Micro. 23 (2) (2003) 22–28.
[7] R. Buyya (Ed.), High-Performance Cluster Computing. Vols. 1 and 2, Prentice Hall, New Jersey, 1999.
[8] O. Celebioglu, R. Rajagopalan, R. Ali, Exploring InfiniBand as an HPC cluster interconnect, (October) (2004).
[9] Cray, Inc, CrayXT System Specifications. www.cray.com/Products/XT/Specifications.aspx, January 2010.
[10] B. Dally, GPU Computing to Exascale and Beyond, Keynote Address, ACM Supercomputing Conference, November 2010.
[11] J. Dongarra, Visit to the National Supercomputer Center in Tianjin, China, Technical Report, University of Tennessee and Oak Ridge National Laboratory, 20 February 2011.
[12] J. Dongarra, Survey of present and future supercomputer architectures and their interconnects, in: International Supercomputer Conference, Heidelberg, Germany, 2004.

[13] K. Hwang, H. Jin, R.S. Ho, Orthogonal striping and mirroring in distributed RAID for I/O-Centric cluster computing, IEEE Trans. Parallel Distrib. Syst. 13 (2) (2002) 26–44.
[14] K. Hwang, Z. Xu, Support of clustering and availability, in: Scalable Parallel Computing, McGraw-Hill, 1998, Chapter 9.
[15] K. Hwang, C.M. Wang, C.L. Wang, Z. Xu, Resource scaling effects on MPP performance: STAP benchmark implications, IEEE Trans. Parallel Distrib. Syst. (May) (1999) 509–527.
[16] K. Hwang, H. Jin, E. Chow, C.L. Wang, Z. Xu, Designing SSI clusters with hierarchical checkpointing and single-I/O space, IEEE Concurrency (January) (1999) 60–69.
[17] H. Jin, K. Hwang, Adaptive sector grouping to reduce false sharing of distributed RAID clusters, J. Clust. Comput. 4 (2) (2001) 133–143.
[18] J. Kevin, et al., Entering the petaflop era: the architecture of performance of Roadrunner, www.c3.lanl.gov/~kei/mypubbib/papers/SC08:Roadrunner.pdf, November 2008.
[19] V. Kindratenko, et al., GPU Clusters for High-Performance Computing, National Center for Supercomputing Applications, University of Illinois at Urban-Champaign, Urbana, IL, 2009.
[20] K. Kopper, The Linux Enterprise Cluster: Building a Highly Available Cluster with Commodity Hardware and Free Software, No Starch Press, San Francisco, CA, 2005.
[21] S.W. Lin, R.W. Lau, K. Hwang, X. Lin, P.Y. Cheung, Adaptive parallel Image rendering on multiprocessors and workstation clusters. IEEE Trans. Parallel Distrib. Syst. 12 (3) (2001) 241–258.
[22] J. Liu, D.K. Panda, et al., Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics, (2003).
[23] R. Lucke, Building Clustered Linux Systems, Prentice Hall, New Jersey, 2005.
[24] E. Marcus, H. Stern, Blueprints for High Availability: Designing Resilient Distributed Systems, Wiley.
[25] TOP500.org. Top-500 World's fastest supercomputers, www.top500.org, November 2010.
[26] G.F. Pfister, In Search of Clusters, second ed., Prentice-Hall, 2001.
[27] N.H. Sun, China's Nebulae Supercomputer, Institute of Computing Technology, Chinese Academy of Sciences, July 2010.
[28] Wikipedia, IBM Roadrunner. http://en.wikipedia.org/wiki/IBM_Roadrunner, 2010, (accessed 10.01.10).
[29] Wikipedia, Tianhe-1. http://en.wikipedia.org/wiki/Tianhe-1, 2011, (accessed 5.02.11).
[30] Wikipedia, MOSIX. http://en.wikipedia.org/wiki/MOSIX, 2011, (accessed 10.02.11).
[31] Wikipedia, CUDA. http://en.wikipedia.org/wiki/CUDA, 2011, (accessed 19.02.11).
[32] K. Wong, M. Franklin, Checkpointing in distributed computing systems, J. Parallel Distrib. Comput. (1996) 67–75.
[33] Z. Xu, K. Hwang, Designing superservers with clusters and commodity components. Annual Advances in Scalable Computing, World Scientific, Singapore, 1999.
[34] Z. Xu, K. Hwang, MPP versus clusters for scalable computing, in: Proceedings of the 2nd IEEE International Symposium on Parallel Architectures, Algorithms, and Networks, June 1996, pp. 117–123.
[35] S. Zhou, LSF: Load Sharing and Batch Queuing Software, Platform Computing Corp., Canada, 1996.

HOMEWORK PROBLEMS

Problem 2.1

Differentiate and exemplify the following terms related to clusters:

a. Compact versus slack clusters
b. Centralized versus decentralized clusters

c. Homogeneous versus heterogeneous clusters
d. Enclosed versus exposed clusters
e. Dedicated versus enterprise clusters

Problem 2.2

This problem refers to the redundancy technique. Assume that when a node fails, it takes 10 seconds to diagnose the fault and another 30 seconds for the workload to be switched over.

a. What is the availability of the cluster if planned downtime is ignored?
b. What is the availability of the cluster if the cluster is taken down one hour per week for maintenance, but one node at a time?

Problem 2.3

This is a research project to evaluate the cluster architectures of four supercomputers built in recent years. Study the details of the No. 1 supercomputer, the Tianhe-1A, which was announced in the Top 500 list released in November 2010. Your study should include the following:

a. Conduct an in-depth evaluation of the Tianhe-1A architecture, hardware components, operating system, software support, parallelizing compilers, packaging, cooling, and new applications.
b. Compare the relative strengths and limitations of the Tianhe-1A with respect to the three case-study systems: the Jaguar, Nebulae, and Roadrunner, studied in Section 2.5. Use tabulations or plot curves, if you find enough benchmark data to conduction the comparison study.

Problem 2.4

This problem consists of two parts related to cluster computing:

1. Define and distinguish among the following terms on scalability:
   a. Scalability over machine size
   b. Scalability over problem size
   c. Resource scalability
   d. Generation scalability
2. Explain the architectural and functional differences among three availability cluster configurations: *hot standby, active takeover*, and *fault-tolerant clusters*. Give two example commercial cluster systems in each availability cluster configuration. Comment on their relative strengths and weaknesses in commercial applications.

Problem 2.5

Distinguish between multiprocessors and multicomputers based on their structures, resource sharing, and interprocessor communications.

a. Explain the differences among UMA, NUMA, COMA, DSM, and NORMA memory models.
b. What are the additional functional features of a cluster that are not found in a conventional network of autonomous computers?
c. What are the advantages of a clustered system over a traditional SMP server?

Problem 2.6

Study the five research virtual cluster projects listed in Table 2.6 and answer the following questions regarding the coverage on COD and Violin experience given in Sections 2.5.3 and 2.5.4:

a. From the viewpoints of dynamic resource provisioning, evaluate the five virtual clusters and discuss their relative strengths and weaknesses based on the open literature.
b. Report on the unique contribution from each of the five virtual cluster projects in terms of the hardware setting, software tools, and experimental environments developed and performance results reported.

Problem 2.7

This problem is related to the use of high-end x86 processors in HPC system construction. Answer the following questions:

a. Referring to the latest Top 500 list of supercomputing systems, list all systems that have used x86 processors. Identify the processor models and key processor characteristics such as number of cores, clock frequency, and projected performance.
b. Some have used GPUs to complement the x86 CPUs. Identify those systems that have procured substantial GPUs. Discuss the roles of GPUs to provide peak or sustained flops per dollar.

Problem 2.8

Assume a sequential computer has 512 MB of main memory and enough disk space. The disk read/write bandwidth for a large data block is 1 MB/second. The following code needs to apply checkpointing:

```
do 1000 iterations
A = foo (C from last iteration)      /* this statement takes 10 minutes */
B = goo (A)                          /* this statement takes 10 minutes */
C = hoo (B)                          /* this statement takes 10 minutes */
end do
```

A, B, and C are arrays of 120 MB each. All other parts of the code, operating system, libraries take, at most, 16 MB of memory. Assume the computer fails exactly once, and the time to restore the computer is ignored.

a. What is the worst-case execution time for the successful completion of the code if checkpointing is performed?
b. What is the worst-case execution time for the successful completion of the code if plain transparent checkpointing is performed?
c. Is it beneficial to use forked checkpointing with (b)?
d. What is the worst-case execution time for the code if user-directed checkpointing is performed? Show the code where user directives are added.
e. What is the worst-case execution time of the code if forked checkpointing is used with (d)?

Problem 2.9

Compare the latest Top 500 list with the Top 500 Green List of HPC systems. Discuss a few top winners and losers in terms of energy efficiency in power and cooling costs. Reveal the green-energy winners' stories and report their special design features, packaging, cooling, and management policies that make them the winners. How different are the ranking orders in the two lists? Discuss their causes and implications based on publicly reported data.

Problem 2.10

This problem is related to processor selection and system interconnects used in building the top three clustered systems with commercial interconnects in the latest Top 500 list.

a. Compare the processors used in these clusters and identify their strengths and weaknesses in terms of potential peak floating-point performance.
b. Compare the commercial interconnects of these three clusters. Discuss their potential performance in terms of their topological properties, network latency, bisection bandwidth, and hardware used.

Problem 2.11

Study Example 2.6 and the original paper [14] reporting the distributed RAID-x architecture and performance results. Answer the following questions with technical justifications or evidence:

a. Explain how the RAID-x system achieved a single I/O address space across distributed disks attached to cluster nodes.
b. Explain the functionality of the cooperative disk drivers (CCDs) implemented in the RAID-x system. Comment on its application requirements and scalability based on current PC architecture, SCSI bus, and SCSI disk technology.
c. Explain why RAID-x has a fault-tolerance capability equal to that of the RAID-5 architecture.
d. Explain the strengths and limitations of RAID-x, compared with other RAID architectures.

Problem 2.12

Study the relevant material in Sections 2.2 and 2.5 and compare the system interconnects of the IBM Blue Gene/L, IBM Roadrunner, and Cray XT5 supercomputers released in the November 2009 Top 500 evaluation. Dig deeper to reveal the details of these systems. These systems may use custom-designed routers in interconnects. Some also use some commercial interconnects and components.

a. Compare the basic routers or switches used in the three system interconnects in terms of technology, chip design, routing scheme, and claimed message-passing performance.
b. Compare the topological properties, network latency, bisection bandwidth, and hardware packaging of the three system interconnects.

Problem 2.13

Study the latest and largest commercial HPC clustered system built by SGI, and report on the cluster architecture in the following technical and benchmark aspects:

a. What is the SGI system model and its specification? Illustrate the cluster architecture with a block diagram and describe the functionality of each building block.

b. Discuss the claimed peak performance and reported sustained performance from SGI.
c. What are the unique hardware, software, networking, or design features that contribute to the claimed performance in Part (b)? Describe or illustrate those system features.

Problem 2.14

Consider in Figure 2.32 a server-client cluster with an active-takeover configuration between two identical servers. The servers share a disk via a SCSI bus. The clients (PCs or workstations) and the Ethernet are fail-free. When a server fails, its workload is switched to the surviving server.

a. Assume that each server has an MTTF of 200 days and an MTTR of five days. The disk has an MTTF of 800 days and an MTTR of 20 days. In addition, each server is shut down for maintenance for one day every week, during which time that server is considered unavailable. Only one server is shut down for maintenance at a time. The failure rates cover both natural failures and scheduled maintenance. The SCSI bus has a failure rate of 2 percent. The servers and the disk fail independently. The disk and SCSI bus have no scheduled shutdown. The client machine will never fail.
1. The servers are considered available if at least one server is available. What is the combined availability of the two servers?
2. In normal operation, the cluster must have the SCSI bus, the disk, and at least one server available simultaneously. What are the possible single points of failure in this cluster?
b. The cluster is considered unacceptable if both servers fail at the same time. Furthermore, the cluster is declared unavailable when either the SCSI bus or the disk is down. Based on the aforementioned conditions, what is the system availability of the entire cluster?
c. Under the aforementioned failure and maintenance conditions, propose an improved architecture to eliminate all single points of failure identified in Part (a).

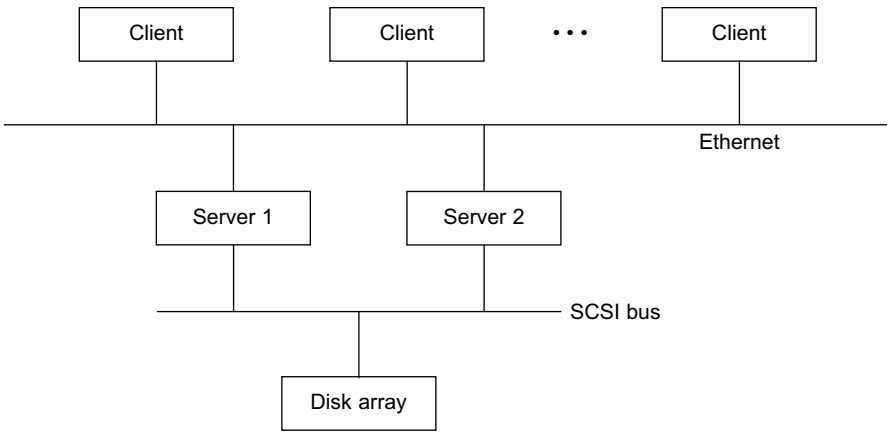


FIGURE 2.32
An HA cluster with redundant hardware components.

Problem 2.15

Study various cluster job scheduling policies in Table 2.6 and answer the following questions. You may need to gather more information from Wikipedia, Google, or other sources if any of the scheduling policies are new to you.

a. Explain the advantages and disadvantages of nonpreemptive and preemptive scheduling policies and suggest methods to amend the problems.
b. Repeat Part (a) for static and dynamic scheduling policies.
c. Repeat Part (a) for dedicated and space-sharing scheduling policies.
d. Compare the relative performance of time-sharing, independent, and gang scheduling policies.
e. Compare the relative performance in stay and migrating policies on local jobs against remote jobs.

Problem 2.16

Study various SSI features and HA support for clusters in Section 2.3 and answer the following questions, providing reasons for your answers. Identify some example cluster systems that are equipped with these features. Comment on their implementation requirements and discuss the operational obstacles to establish each SSI feature in a cluster system.

a. Single entry point in a cluster environment
b. Single memory space in a cluster system
c. Single file hierarchy in a cluster system
d. Single I/O space in a cluster system
e. Single network space in a cluster system
f. Single networking in a cluster system
g. Single point of control in a cluster system
h. Single job management in a cluster system
i. Single user interface in a cluster system
j. Single process space in a cluster system

Problem 2.17

Use examples to explain the following terms on cluster job management systems.

a. Serial jobs versus parallel jobs
b. Batch jobs versus interactive jobs
c. Cluster jobs versus foreign (local) jobs
d. Cluster processes, local processes, and kernel processes
e. Dedicated mode, space-sharing mode, and timesharing mode
f. Independent scheduling versus gang scheduling

Problem 2.18

This problem focuses on the concept of LSF:

a. Give an example of each of the four types of LSF jobs.
b. For a 1,000-server cluster, give two reasons why the LSF load-sharing policy is better if (1) the entire cluster has one master LIM or (2) all LIMs are masters.

c. In the LSF master-election scheme, a node in the "no master" state waits for a time period proportional to the node number before becoming a new master. Why is the wait time proportional to the node number?

Problem 2.19

This problem is related to the use of MOSIX for cluster computing. Check with the open literature on current features that have been claimed by designers and developers in supporting Linux clusters, GPU clusters, multiclusters, and even virtualized clouds. Discuss the advantages and shortcomings from the user's perspective.

Problem 2.20

Compare China's Tianhe-1A with the Cray Jaguar in terms of their relative strengths and weaknesses in architecture design, resource management, software environment, and reported applications. You may need to conduct some research to find the latest developments regarding these systems. Justify your assessment with reasoning and evidential information.